# On the Computation of Common Subsumers in Description Logics

**Dissertation**

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Inform. Anni-Yasmin Turhan
geboren am 12. Februar 1973 in Hamburg

verteidigt am 8. Oktober 2007

Gutachter:
Prof. Dr.-Ing. Franz Baader,
Technische Universität Dresden
Prof. Ian Horrocks,
Oxford University
Prof. Dr. rer. nat. habil. Ralf Möller,
Technische Universität Hamburg-Harburg

Dresden im Mai 2008

# Contents

# Chapter 1

# Introduction

Description Logic systems are a form of knowledge representation and reasoning systems that allow to represent and to reason about conceptual, i.e., terminological knowledge. The representations as well as the reasoning services are based on well-defined formal semantics. The main ingredients for representing terminological knowledge are concept descriptions. For example, such a concept description can characterize the category of 'mother' as a female person who has a person as a child, in the following way:

$$\mathsf{Person} \sqcap \mathsf{Female} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Person}.$$

In this expression Person and Female are concepts and has-child is a role – a binary relation.

In this thesis we devise methods to infer generalizations of concept descriptions. More precisely, we discuss methods that extend well investigated methods for computing *least common subsumers* of a collection of concept descriptions to more expressive Description Logics, provide first implementations of the most promising techniques and discuss techniques to improve these reasoning services. The methods here investigated can provide assistance for users that are not experts in knowledge representation to build and extend their knowledge bases. We start this endeavor with a short overview of Description Logics systems.

## 1.1 Knowledge representation with Description Logics

Description Logic systems typically consist of three components. First, there is the representation of basic categories of the domain of interest. These categories constitute the *terminological knowledge* about the application domain. This information is represented by *concept descriptions*, which are built from concept constructors and concept names, some of which were used in the example above. Names can be assigned to these concept descriptions in *concept definitions*, e.g.,

$$\begin{aligned} \mathsf{Mother} &\equiv \mathsf{Person} \sqcap \mathsf{Female} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Person} \\ \mathsf{GrandMother} &\equiv \mathsf{Mother} \sqcap \exists \mathsf{has\text{-}child}.\exists \mathsf{has\text{-}child}.\mathsf{Person} \end{aligned}$$

The concept Mother is used to define other concepts in turn. The set of concept definitions is stored in the *TBox* (terminological box) or *ontology*.

The second component of a description logic system is the *ABox* (assertional box). Here, information about concrete entities from the application domain is asserted. The individuals are described again by concept descriptions and by relations to other individuals. For example, we can state in an ABox that Nelli and Anne are female persons, Anne has at least two children Anne has the child Nelli and Nelli has a brother Sammy in the following way:

$$\begin{aligned} \mathsf{Person} \sqcap \mathsf{Female} \sqcap (\geq\ 2\ \mathsf{has\text{-}child}) \quad & (\mathsf{Anne}), \\ \mathsf{Person} \sqcap \mathsf{Female} \quad & (\mathsf{Nelli}), \\ (\mathsf{Anne}, \mathsf{Nelli}) \ : \ & \mathsf{has\text{-}child}, \\ (\mathsf{Nelli}, \mathsf{Sammy}) \ : \ & \mathsf{has\text{-}brother} \end{aligned}$$

The ABox typically refers to concepts defined in the TBox—as in case of the concept Person in our example. The TBox and the ABox together are also referred to as the *knowledge base*.

The third part of description logic systems is the reasoning component. Based on the explicit information given in TBox and ABox, the reasoning component can infer implicit facts automatically. One of the typical reasoning services is to test whether the information captured in TBox and ABox contains a contradiction. This is not the case in our above example. But, for instance, the DL system can infer from the information given that Anne is a mother. This is not stated explicitly, but since Anne is a female person and she has a child, the description of her also fulfills the description of the concept Mother from the TBox.[1]

### 1.1.1 Description Logics

Early knowledge representation systems such as *semantic networks* [Qui67; Sow91] or *frame systems* [Min74] are predecessors of description logics systems. These knowledge representation systems were motivated by cognitive science, more precisely by linguistic applications. These kinds of systems allow to specify information on notions from the domain of discourse and offer methods to compute inheritance relation between the specified notions. Semantic networks are graphical representation of notions about the domain of discourse. Nodes in these graphs represent concepts or individuals and the labeled edges between them represented the relations between them. In frame systems, concepts are represented by *frames*, which are a record-like structure. A frame has a name for the concept it describes, a set of super-concepts and a set of *slots*. Each slot describes a property of the concept by linking to other frames. Both frame-based systems and semantic networks have operational semantics, i.e., the semantics of reasoning is 'defined' by the implementation of the reasoning task. As a consequence the result of the reasoning process depends on the implementation of the reasoner and thus the result may differ from system to system for the same input [Sow92]. To remedy this, Description Logics (DL) are based on formal semantics

[1] The notions used in this Section will be introduced formally in Section 2.

and their reasoning tasks are defined based on these semantics. The information on the domain of discourse is represented in a declarative and unambiguous way. The formal semantics of the reasoning tasks ensures predictable and reliable behavior of DL reasoning systems independent of the implementation.

Early DL research on reasoning methods started from DLs that offered only a small range of concept constructors. The employed reasoning algorithms were mainly *structural algorithms*. These algorithms typically proceed in two phases: first the concepts are normalized and then the structures of the normalized concepts are compared. It turned out that already for rather small DLs the complexity of reasoning was not tractable [Neb88; LB87; DLNN91]. From then on research in the fields of DLs was mainly devoted to finding sound and complete reasoning algorithms for more expressive DLs that offered a good trade-off between expressiveness and tractability.

Most Description Logics are fragments of first order logic. Many of the most common concept constructors can be translated into the two-variable fragment of first order logic [SCM03; LSW01; HSG04]. DLs are closely related to Modal Logics [BdRV01]. In [Sch91] it was shown that the DL $\mathcal{ALC}$ is a notational variant of the Modal Logic $\mathbf{K_m}$. Many theoretical results on reasoning in modal logics carry directly over to standard inferences in DLs.

### 1.1.2 Standard inferences in Description Logic systems

From the information that is captured explicitly in the TBox and ABox, more information can be implied. To be able to infer implicit information automatically is one of the main assets of DL systems. DL systems implement different methods to infer knowledge implicitly captured in the knowledge base. We refer to these typical and well investigated inferences as *standard inferences*. One of the basic standard inferences is to test for *satisfiability* of the TBox, i.e., testing whether the information specified there is not contradictory. In case the information is contradictory, any consequence follows logically from the TBox, which is obviously an undesired effect. Moreover, since it is unsatisfiable, the specified information does not capture the intended notions from the real-world domain. To test for satisfiability is often a first step for the knowledge engineer to check whether the knowledge base models something meaningful.

Another typical standard inference offered in DL systems is to compute whether one concept in the TBox models a more general category than another one. In that case the concept *subsumes* the other concept. *Subsumption* is the standard inference to infer super- and sub-concept relationships between concepts. To compute the subsumption relations for all concepts mentioned in the TBox is called *classification* and yields the *concept hierarchy* or *taxonomy* of the TBox. By computing the subsumption hierarchy the knowledge engineer can compare if the obtained concept hierarchy matches her knowledge about the domain.

From some information stated in the ABox it might follow implicitly that some individuals are instances of concepts defined in the TBox, although not explicitly mentioned. The inference *instance checking* computes for an individual and a concept whether this individual is an instance of the given concept w.r.t. the knowledge base.

This inference helps the knowledge engineer to assess whether the intended relationships between individual descriptions and concepts are captured in the knowledge base or whether these descriptions need further refinement.

These are only some of the typical reasoning services offered by DL systems, for a broader overview see [BN03]. We call the reasoning services that refer to concepts and TBoxes *terminological reasoning* and those that refer to ABoxes in addition *assertional reasoning*. In this thesis we focus on reasoning about concepts and will mainly discuss terminological reasoning.

All of the above mentioned inferences have been investigated for a great range of DLs. In particular, tableaux-based decision procedures for these inferences have been devised and their complexities have been studied extensively [BS01]. For instance, the complexity of testing for subsumption between two concepts w.r.t. a TBox written in the already mentioned DL $\mathcal{ALC}$ is PSPACE-complete [SS91]. The worst case complexity of subsumption in DLs that are nowadays widely used as ontology languages is—as for example the DL called $\mathcal{SHIF}$—EXPTIME-complete or for $\mathcal{SHOIQ}$ even NEXPTIME-complete [Tob01; HS05]. Already due to early findings reasoning in DLs was considered to be intractable [Neb90] and thus not feasible for practical applications. However, it was unclear whether these high worst case complexities would be encountered in practice.

In the late nineties tableaux reasoning methods for these DLs were implemented in a highly optimized way. These implementations used some well-known techniques to speed-up reasoning [DLL62; Bak95], but, more importantly, new techniques tailored to DL reasoning methods were developed [BFH+94; Hor97; HST00; HMT01]. These techniques were first put to practice in the DL reasoner system FACT [Hor98] and then in the system RACER [HM01b]. It turned out that these highly optimized implementations of the terminological reasoning methods do perform surprisingly well on TBoxes from practical applications. Classification of big and complex TBoxes for expressive DLs was possible within minutes. It also turned out that those cases that lead to the worst case complexities of the reasoning methods are not often encountered in practical applications [Hor98; HPS99; HST00; HM01a; Hor02; THPS07]. These findings encouraged research on standard reasoning methods for even more expressive DLs and on how to implement these methods efficiently. This led to a collection of implementations of DL reasoners, such as FACT++ [TH06], RACERPRO [Rac05], PELLET [SP04], KAON2 [Mot06], CEL [BLS06] and most recently HERMIT [MSH07]. While the first three reasoners are tableaux-based systems, the latter three investigate alternative reasoning techniques.

Although today's DL standard reasoners are highly optimized for TBox reasoning, they are not yet capable of handling ABox reasoning for very large knowledge bases efficiently. High-performance ABox reasoning is an ongoing research issue [HM04; HMW05]. For many practical applications DL systems and their standard inferences in particular are employed successfully.

### 1.1.3 Practical applications of Description Logics

Next, we give an overview of some common applications of DL systems and the knowledge bases build for these applications to address the issue of how users can be assisted in building knowledge bases suiting the needs of their application.

#### Natural Language Processing

The motivation for early DL systems was their practical application in *natural language processing*. In this application area, methods from knowledge representation are used to analyze and interpret, but also to generate sentences in natural language. Knowledge bases are built to model syntactic constituents of sentences—such as *noun phrases* or *verb phrases*—and also contextual knowledge and knowledge about the specific domain of discourse. When applied in natural language interpretation, these knowledge bases help to limit the number of possible interpretations of an examined utterance or sentence, since the semantically implausible interpretations can be ruled out early in the interpretation process [Fra03].

A good linguistic knowledge base should be usable for both tasks—natural language understanding as well as for natural language generation. Furthermore, good knowledge bases for natural language processing can be divided into an *upper ontology* and a *domain ontology* [Bat90; BMF95; KL94]. The upper ontology models the language-dependent, but domain-independent parts, while the domain ontology captures notions from the domain of discourse. This aspect should be taken into account when building this kind of knowledge bases.

#### Chemical Process Engineering

In *chemical process engineering*, DL systems are used to build knowledge bases that capture the compositions of complex systems and their building blocks. Process engineering is concerned with methods, tools, and their management for the analysis, design and control of a process. Here, *models* are used to represent, analyze, and optimize processes and get a deeper understanding of their nature. In particular, equation-based mathematical models are desirable, because of their high predictive capabilities in numerical analysis and simulation. Unfortunately, these models are very complex even for simple chemical processes. Nevertheless, adequate models can be obtained step by step, starting with so-called *block-oriented* models. Each *block* stands for a standardized sub-unit of the entire process with certain interfaces and each *connection* for a flow of material, energy, or information. Typically, block-oriented modeling environments have a *block repository* in which building blocks and their composites are stored. In case a DL system is used as a block repository, building blocks and their partonomies are modeled in the TBox and individual system parts are modeled in the ABox.

The ontologies for process engineering are structured in a layered way according the generality of the described concepts [BLM01; MYM07]. This design of the ontologies facilitates the navigation in the ontology for the process engineer when searching for a particular building block. More importantly, this design aims at the re-usability of

the ontology in different process engineering applications, since it offers descriptions of building blocks at the 'right' level of detail and granularity. For example, in the chemical process engineering ontology OntoCAPE[2] developed at the RWTH Aachen, five different abstraction layers are identified: meta layer, upper layer, conceptual layer, application-oriented layer and application-specific layer [MYM07].

Many requirements that chemical process engineering applications pose to knowledge representation systems, such as descriptions of building blocks at different abstraction levels, consistency tests for the modeled block descriptions and automatic classification of the repository can naturally be fulfilled by DL systems as the in-depth investigation in [Sat98] showed.

### Life Sciences

The most active application area of DL systems are currently the *life sciences*, which comprises the wider fields of biology and medicine. Here, the modeling of the domain in a formal representation language is a benefit in itself, since this formalizes and to some extent standardizes what the community understands about certain terms in an unambiguous way. Thus by agreeing upon an ontology and on the definition of concepts in it, a community can create a shared understanding of their domain of study as noted in [WLT+06]. The obtained ontologies simply serve as a community knowledge reference and thereby reduce heterogeneity in the community. This effect has been demonstrated prominently by the Gene Ontology [Con00], where several ontologies use the same terminology to describe the major attributes of functionality of gene products by a controlled vocabulary. This particular ontology was obtained from a joint effort of different genome projects [BER+98; Con98]. It contains about 18.800 concepts. Furthermore, DL systems are applied to check the consistency of the modeled information.

In health care knowledge bases are used for decision support and also for semantics oriented natural language processing of medical reports [Rec03]. In the health care domain ontologies have been developed for many years and a range of medical ontologies have been developed; we take a closer look at two of them.

The work on early precursors of the Snomed[3] ontology has started already in the seventies [CRP+93]. It models anatomical structures, diseases and medical procedures. Today the current version of Snomed ontology is applied for health reporting, billing and statistical analysis in USA and throughout Europe. The DL version of the Snomed Ontology describes more than 397,000 concepts and more than 50 roles, but it uses only a small number of concept constructors to describe the concepts [BLS07].

The European Galen project started in the early nineties to develop a common reference model for the integration of medical information systems [RNG93; Rec03]. The resulting Galen ontology uses a very expressive DL and moreover *general concept inclusion axioms* (GCIs), which state subsumption not between a concept name and a complex concept description, but between two concept descriptions. The Galen ontology contains about 2,700 concept definitions and 1,200 GCIs [BLS07].

[2]CAPE stands here for Computer-Aided Process Engineering.
[3]Snomed: 'Systematized nomenclature of medicine'

In medicine as well as in biology ontologies can grow extremely large. Obviously, these ontologies are not developed by a single person, but by a whole team of persons and mostly these ontologies are developed over a long period of time. All of these circumstances necessitate automated support for constructing and maintaining knowledge bases from this domain.

**Semantic Web**

In the last couple of years the application area of the *semantic web* [BLHL01] brought more attention to DLs and their powerful reasoning systems. The semantic web is envisioned as a future version of today's World Wide Web, where web content will be annotated with formal representation of its meaning. This formal representation is then used and interpreted by applications to automatically find and choose useful web resources for their tasks.

The annotation can state in which context a keyword is used. For a web page with the title 'Java guide' the annotation can clarify that the Indonesian island and thus not the programming language is referred to. In case this annotation is done w.r.t. an ontology and if it can be inferred from this ontology that Java is a part of Indonesia, this page is returned to the general request for information on Indonesian travel destinations. This kind of result would not be found with the syntax-based search techniques employed in search engines today. Similar to this, other kinds of resources can be annotated with ontologies to be used in the semantic web, as for example web services.

Although mainly a future vision, the semantic web is already a strong motivation for the development of powerful reasoning algorithms for very expressive DLs on the one hand and for the implementation of DL systems and DL tools on the other. An important step towards realizing the semantic web was the standardization of ontology languages such as DAML+OIL [CvHH$^+$01] and, more importantly, of the web ontology language OWL [BvHH$^+$04; HPSvH03]. The W3C recommendation for OWL specifies three dialects. While the most expressive dialect *OWL full* is beyond the expressivity of DLs and reasoning in it is undecidable, the other two dialects correspond to DLs for which sound and complete reasoning procedures exist. *OWL DL* can express ontologies written in the DL $\mathcal{SHOIN}$ and the less expressive *OWL lite* can express ontologies written in the DL $\mathcal{SHIF}$.

The close relation of DLs and these two OWL dialects raised the interest of new user groups in DL reasoning and DL systems. These users come from various fields and do not always have a background in knowledge representation or logics. Thus, it is often hard for these users to understand what has been specified in the knowledge base, since large and complex concept descriptions are not easy to comprehend. The more severe problem for inexperienced users is to capture their intuitive notion of a concept in a formalism like DLs or OWL. To provide automated support for these users in building and extending their knowledge bases in a meaningful way is a requirement for ontology tools for the semantic web. Furthermore, ontologies for the semantic web are likely not to be built from scratch, but to re-use ontologies or parts of them from other applications. Thus tools for editing ontologies should also facilitate the

comprehension of the structure of knowledge bases and the meaning of the concepts therein.

These different application areas of DL systems show that knowledge bases for practical applications are mostly developed by more than one person and often by non-logicians. Furthermore, the ontologies from real applications can grow very large and complex. These issues make it desirable to have automated support for ontology users for the tasks of design and maintenance of Description Logic knowledge bases.

## 1.2 Design and maintenance of Description Logic knowledge bases

To identify tasks for design and maintenance for DL knowledge bases, we first summarize basic principles for good ontology design and then derive some demands for good support for ontology design and maintenance from the practical applications of DL systems that we just described. Gruber established some basic principles for good ontology design for formal ontologies in [Gru93].

**Clarity.** An ontology should state the intended meaning of terms in an objective way. The meaning captured should be dissociated from the social and computational context. A complete definition giving the necessary and sufficient conditions is to be preferred over incomplete ones.

**Coherence.** The definitions in an ontology should be logically consistent, neither the definitions of terms nor their individuals should be contradictory.

**Extendibility.** The structure and the definitions in an ontology should allow to extend the ontology monotonically. More precisely, the definition of new terms should preserve the meaning of the terms they refine.

**Minimal encoding bias.** An ontology should be specified at the knowledge level, instead of choosing a format to accommodate convenience of notation or implementation.

**Minimal ontological commitment.** An ontology should make as few claims as possible about the modeled domain, i.e., only the terms essential for the intended use of the ontology should be defined. Ontological commitment can be minimized by specifying the weakest theory and thereby allowing the most models.

Some of these quality criteria are met by DL systems quite naturally. For instance the criteria clarity and minimal encoding bias are supported by the formal semantics of DLs. Coherence of the ontology can be tested by standard reasoning methods implemented in DL systems. The minimal encoding bias can be diminished in modern ontology editors and visualization tools that offer an abstract syntax for writing down the ontology. The user does not deal explicitly with the underlying format of the ontology such as RDF, but only with human readable formats and thus does not have

to compensate for the format when modeling. However, some of the above criteria can be met by employing new reasoning tasks, while others are simply requirements from the mentioned practical applications for good ontology design which we discuss next.

**Avoid redundancy.** Ontologies that contain redundancies are unnecessarily large and often not well-structured. This makes it harder for the modeler to comprehend the ontology and, more importantly, to extend the ontology. Furthermore, redundancies can lead to unexpected reasoning results. The cause for such a result is more difficult to locate and to repair in redundant ontologies.[4]

**Re-usable ontologies.** Many application domains re-use knowledge bases that model a sub-domain of their domain of interest. For instance, in the process engineering domain basic building blocks are re-used in other ontologies, while in the area of the semantic web upper ontologies are often used to model general domains that are referred to in the application knowledge base.

**Levels of granularity and abstraction.** Ontologies that model different levels of granularity and abstraction are easier to extend. To this end ontologies in the domain of life sciences and also in the process engineering domain ontologies are designed in conceptual layers.

**Balanced concept hierarchy.** By balanced concept hierarchy we mean a concept hierarchy that does not have a shallow hierarchy and where a node only has a 'reasonable' number of children. Balanced hierarchies facilitate the comprehension of the (structure of) the ontology. The concept hierarchy is the structure used for browsing the ontology. For instance, if the process engineer looks for a particular building block, a balanced concept hierarchy enables better navigation in large ontologies.

In order to propose methods to meet these requirements for good ontology design one has to bear in mind under which conditions ontologies are developed. Firstly, many ontologies have been developed over a period of many years. For instance the medical ontologies Galen and Snomed have been developed over at least a decade. Secondly, large ontologies are not developed by a single person, but by a whole team of modelers. Thirdly, most application ontologies are developed by users who are experts in the domain that they model, but not in the representation formalism they use.[5] It is evident that in this kind of setting support for the design and maintenance of ontologies is needed. On the one hand a useful ontology system should facilitate the comprehension of the knowledge base to help a naive user to get an overview of the ontology and to see how the domain is modeled in order to extend or re-use an ontology in a competent way. On the other hand an ontology system should offer support for meeting the design requirements. However, this support must be based on well-defined methods to guarantee predictable outcome and reliable behavior of the

[4] This design goal complements the criterion of clarity proposed by Gruber.

[5] In that sense we speak of 'naive users' to refer to this group of DL system users.

system. For most of the above mentioned requirements inference services have been proposed that assist naive users in accomplishing the tasks of ontology design and maintenance.

### 1.2.1 Reasoning support for design and maintenance of knowledge bases

The inferences satisfiability or subsumption can be used to test the quality criterion of coherence of a knowledge base or to test whether the intended super-concept relations are implied by the ontology. Now, in case the result of these inferences is not as expected, it is often not easy to see why and, more importantly, which statements in the ontology cause these results. It is crucial for the modeler to identify these statements and to change the ontology such that the intended subsumption relationships hold. The identification of such relevant statements can be achieved by the task of *axiom pinpointing*. Axiom pinpointing is investigated for expressive DLs in [PSK05; BP07] and for the small but lightweight DL $\mathcal{EL}$++ [BPS07].

The re-use of ontologies requires that the modeler comprehends the ontology written by others that is considered as a candidate for re-use. In order to be able to extend the ontology in a meaningful way, the user must understand the definitions stored in the ontology. For very expressive DLs the user might be overwhelmed by very complex definitions and not be able to assess in what kind of effects changes would result. To obtain a version of the ontology where fewer concept constructors are used—that is the closest version w.r.t. subsumption to the original ontology—*concept approximation* has been proposed. We formally introduce and examine this inference in Chapter 4. Concept approximation has been investigated in [BKT02b] for the first time. The idea is to eliminate unwanted concept constructors by computing the closest concept description in a DL where this concept constructor is not present. In particular, disjunction is often confusing for naive users. In so-called frame-based views of ontologies, disjunction is not supported. Now, one can use concept approximation to obtain a frame-based view from an ontology written in a more expressive DL.

When aiming at the re-usability of an ontology, a natural question is whether a sub-part of the ontology can be identified that contains the necessary information to retain the subsumption hierarchy of the concepts that appear in this sub-part of the ontology, if this sub-part is classified alone. Such a sub-part is called an *ontology module* and constitutes a part of the ontology that is somewhat independent of the rest of the ontology and can be exported to other ontologies while preserving its concept hierarchy. The identification of such ontology modules has been investigated in [CKHS07; CHKS07], where sufficient syntactic conditions have been devised to identify ontology modules. These conditions can test whether modules in a $\mathcal{SHOIQ}$-ontology exist and are currently being implemented to complement the services offered by the ontology editor SWOOP [KPS+06].

In regard of extensibility and re-use of an ontology or parts of it another question is: Does importing it affect my existing ontology in an unwanted way? For instance, is the concept hierarchy of the 'old concept names' the same after the import? To ensure

this, *conservative extensions* have been devised in [GLW06]. This reasoning service decides whether it is safe to add a set of ontology statements to an existing ontology, i.e., if it does not change the concept hierarchy of the older part. The methods for recognizing conservative extensions have been so far investigated for $\mathcal{ALC}$ [GLW06] and $\mathcal{ALC}$ extended by qualified number restrictions and inverse roles [LWW07].

The design requirements of modeling different levels of granularity as well as a balanced concept hierarchy can both be achieved by the introduction of concepts that are intermediate concepts, i.e., concepts that are located in the taxonomy between a concept and its children. To this end such a concept has to be more general than the children concepts and be more specific than the initial parent concept. The inference that yields generalizations of a collection of concepts is the *least common subsumer*. More precisely, this inference yields a concept description that expresses the commonalities of all input concepts. This concept description can in turn be the starting point for a new concept definition in the ontology. By applying the computation of the least common subsumer to similar concepts that have high level of refinement in regard of the application domain, the modeler can directly obtain a concept description that describes a more abstract notion of the application domain.

The layered design brings us to the more general issue of how the extension of ontologies can be supported in a semi-automatic way. In DL systems standard inferences are employed when a concept description is already written down—little support is given to come up with a concept description matching the intuition of the modeler and capturing certain characteristics of the notion in mind. For instance, the OWL plug-in [KMR04] of the ontology editor PROTÉGÉ [GMF+03] offers *modeling patterns*. One of these patterns for example guides the user in a step-by-step fashion to specify a concept as a partition of a set of other concepts. However, still the user has to bear in mind which aspects of the concept have to be addressed in the definition and specialized. Some approaches that provide the user with a concept description as a starting point for editing and that take the structure of the knowledge base into account are based on non-standard inferences.

### 1.2.2 Approaches for extending ontologies based on non-standard inferences

Typically, Description Logic knowledge bases are constructed in *top-down* fashion: first, the knowledge engineer formalizes basic categories of the domain as concepts in the TBox, which are then refined. In a later step concrete entities from the domain are described by individuals in the ABox. We describe four scenarios for extending existing ontologies that can be realized by the use of so called non-standard inferences.

#### Bottom-up extension

The modeler starts from individuals in the ABox to introduce new concepts in the TBox. In a first step the knowledge engineer selects a set of individuals modeled in the application ABox that, in her intuition, share common characteristics and are prototypical for the category the modeler has in mind. From this set of selected

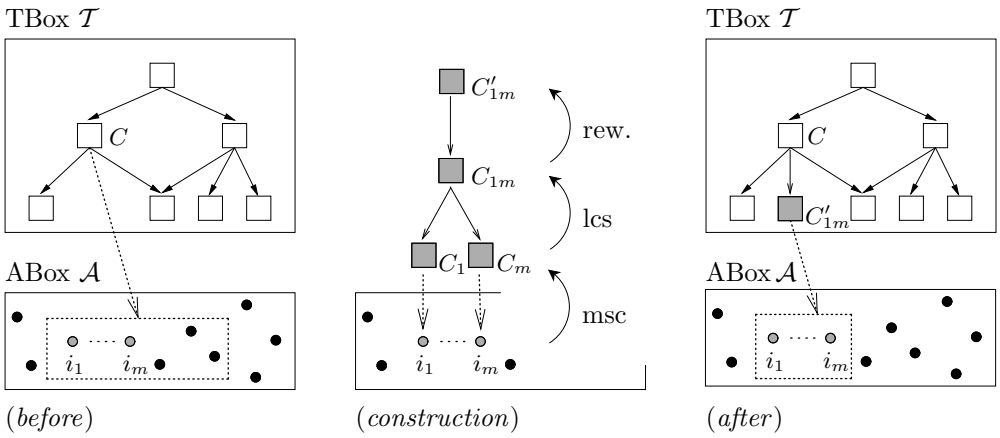


Figure 1.1: Bottom-up extension of knowledge bases.

individuals the DL system generates automatically a concept description that is the most specific one of which all of the picked individuals are instances. The concept description is then offered to the modeler for inspection and editing. If the modeler is confident with the concept description, she can assign a name to it and add this concept definition to the TBox.

This way of extending knowledge bases has been practiced 'by hand', i.e., without the automated support in the domain of process engineering and was in fact the motivation for earlier studies on non-standard inferences that realize this approach [BKM99; Mol00]. Recently, this way of supporting users has been demanded in [KRM07] for the building of bio-ontologies.

In Figure 1.1 the realization of the bottom-up extension is depicted. On the left side of the figure the taxonomy of the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ are shown. The concept $C$ in $\mathcal{T}$ has a number of instances in the ABox (pointed to by the dotted arrow). Now, the modeler wants to introduce a sub-concept of $C$ that has the individuals $i_1, \ldots, i_m$ as instances. The bottom-up approach typically involves three non-standard inferences depicted in the middle of Figure 1.1. First, for each of the individuals selected by the modeler the *most specific concept (msc)* is computed. From a description of an individual and its relations to other individuals specified in the ABox, this inference computes a concept description that is the most specific concept description of which all selected individuals are instances of. In the example in Figure 1.1 the concept description $C_1$ to $C_m$ are the most specific concepts of the selected individuals. Next, in the bottom-up extension, the obtained concept descriptions are generalized into a single concept description by computing their *least common subsumer (lcs)*. The lcs is a concept description that is the closest w.r.t. to subsumption to the concept descriptions it is applied to. In our case the lcs of the descriptions obtained in step one yields $C_{1m}$. As we will see, the concept descriptions obtained by computing the lcs can grow very large. Thus it is often useful to compute the *minimal rewriting* of the concept description obtained by the lcs. This rewriting step returns an equivalent concept that is more compact than the original one. It proceeds by finding parts

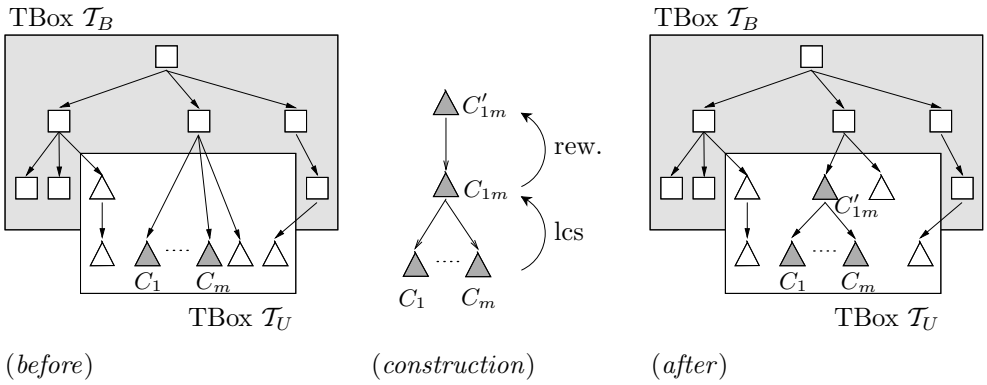


Figure 1.2: Customization of Background Ontologies.

in the concept description that can be replaced by a concept name from the TBox while preserving equivalence. This step yields concept description $C'_{1m}$. This concept description can be assigned a name and added to the TBox. In the right part of Figure 1.1 we see the concept hierarchy extended by the automatically constructed concept $C'_{1m}$, which has the initially chosen individuals as instances.

A variation of the bottom-up extension, which we will study in this thesis instead of the one described above, starts from a selection of concepts from the TBox and provides a concept description obtained by applying the lcs as a candidate for a new concept to be introduced in the TBox.

**Customization of background ontologies**

This method for extending knowledge bases operates with two ontologies written in different DLs. Suppose the modeler has little expertise in DLs and thus uses an easy to comprehend, but rather inexpressive DL to build her ontology. Furthermore for modeling her application domain she needs to make reference to notions formalized in an ontology written in a more expressive DL. We call this ontology the *background ontology*. Perhaps the background ontology is a standardized ontology and its original definitions must be used or it is obtained from an ontology vendor and the concept definitions must not be changed. So, the modeler uses concept names from this background ontology in the user ontology to define new concepts and to adapt the concepts defined in the background ontology to her application. This approach is displayed in Figure 1.2 above. Here the concept hierarchy of the background terminology $\mathcal{T}_B$ and the user terminology $\mathcal{T}_U$ is depicted. The concepts from the different ontologies are depicted by different geometrical shapes, since they are written in different DLs.

Now, in order to customize the background ontology by a well-structured concept hierarchy, the modeler chooses similar concepts that form a category from the user ontology for which a super-concept should be obtained—in Figure 1.2 these concepts are named $C_1$ to $C_m$. The DL system generalizes these concepts into one concept description ($C_{1m}$), compresses it by rewriting ($C'_{1m}$) and returns it for inspection and manual editing to the modeler. After the concept is assigned a name, the resulting

concept hierarchy looks as in the right part of Figure 1.2, where the taxonomy of $\mathcal{T}_B$ and $\mathcal{T}_U$ has an additional level.

Customization of knowledge bases is realized by the *least common subsumer* w.r.t. the combined ontology. The obtained lcs concept description uses only the *concept constructors* present in the user DL, but *concept names* from the background ontology might also be used. In some cases where the lcs is difficult to obtain, a relaxed version of the lcs might be useful for efficiency reasons. To this end we propose to compute a *good common subsumer*, which also subsumes all input concept descriptions, but is not necessarily the least concept description to do so.

This approach for extending knowledge bases by customization was initially proposed in [BST04a]. We will investigate the inferences supporting customization of background ontologies in Section 5.

#### Extension by import

Similar to the case of customization of background ontologies, the modeler uses a rather inexpressive DL to build her ontology and, again, wants to use concepts that are defined in another ontology written in a more expressive DL. But this time the user wants to import the concept definitions directly into her own ontology. In order to avoid increasing the expressivity of the ontology—for instance to keep nice computational properties of reasoning in this DL—the concept descriptions of the concepts to be added have to be 'translated' into the less expressive DL first.

This translation is realized by the non-standard inference *concept approximation*, which was introduced in [BKT02b]. Concept approximation of a concept description in the expressive *source DL* computes the closest (w.r.t. subsumption) concept description in the less expressive *destination DL*. The name of the inference stems from the fact that not all of the information captured in the initial concept can be expressed in the less expressive destination DL and some information might be lost.

Sometimes it might be useful to translate a whole ontology to a less expressive DL. For instance, if certain reasoning services are only available in this destination DL. In this case concept approximation can be employed to get the closest version (w.r.t. subsumption) of the original ontology in the DL supporting the inference.

#### Extension by modification

Consider the situation where the modeler has to build many similar concepts. Instead of starting from scratch to write down the concept description, it is a better starting point to find a concept description in the ontology that is *structurally* similar to the intended one and modify it. Now, in order to find such a concept, the knowledge engineer needs to specify only some parts of the concept description and leave some parts open—this can be done by concept patterns. Intuitively, *concept patterns* are structured as concept descriptions for which in addition *concept variables* can be used in the place of concept names.[6] Based on concept patterns the search for a structurally similar concept can be solved by the inference of *concept matching*. A

[6]Except for the fact that the primitive negation ($\neg$) may not occur in front of variables.

*matching problem* for a DL $\mathcal{L}$, a concept pattern $P$ and a concept description $C$ asks whether the variables in $P$ can be instantiated with concept descriptions in $\mathcal{L}$ s.t. the obtained concept description is equivalent to $C$. Intuitively, if a concept can be matched against a pattern $P$, then their syntax trees share the 'upper part', where $P$ is fully specified. Deviations may occur at leaves labeled with variables. In our application it is interesting to obtain those concepts defined in the TBox that match the pattern.

This way of extending ontologies was described for the application of chemical process engineering in [BT01b]. An extensive overview for and recent results on concept matching are presented in [Bra06].

#### Extension by completion

Suppose the modeler has written down the TBox and the ABox and wants to ensure whether the knowledge base contains all the relevant information about the application domain and, if information is missing, wants to extend the knowledge base appropriately. In [BGSS07; Ser07] an interactive method was proposed that aims at completing the knowledge base—TBox and ABox—by asking the domain expert questions such as:

- Are all the relationships that hold among the already introduced concepts captured by the constraints in the TBox? Or are there relationships that hold in the domain, but do not follow from the TBox?
- Are all kinds of instances from the application domain represented by individuals in the ABox? Or are there instances in the domain that have not yet been included in the ontology or even not yet been identified?

In case the answer is negative, the domain expert is asked to supply a counterexample.

The method underlying this approach is *Formal Concept Analysis* (FCA) [GW99]. Formal concept analysis has been introduced as a method of data analysis based on so-called *formal contexts*. A formal context consists of a pair of a set of objects and a set of attributes. In its simplest form a formal context is a way of specifying which objects are satisfy which attributes. Usually, formal contexts represent complete knowledge of the domain. Due to the open world semantics of DL systems, the notion of a context had to be adapted to *partial contexts* in [BGSS07].

Now, *attribute exploration* [GW99] is a knowledge acquisition method of FCA that is used to acquire knowledge from a domain expert by asking questions successively. In many applications the formal context is not explicitly given, but it is rather 'known' to a domain expert. The system presents hypotheses about the concepts to the expert, which the expert has to accept or reject by supplying a counterexample. Intuitively, the methods from FCA and attribute exploration in particular, ensure for the above mentioned application that the interaction with the expert is kept to a minimum and, moreover, that the knowledge base is completed in a well-defined sense. In [BGSS07] a notion of completeness of a knowledge base is introduced w.r.t. a fixed model. The approach of extension by completion has been implemented as a extension to the

ontology editor SWOOP (see [Ser07]) and is currently being evaluated w.r.t. a bio-ontology on human protein phosphatases [WBH+05].

We focus on the first two approaches for extensions in this thesis and will study concept approximation—the central inference for the third approach in Section 4. The fourth approach is discussed in detail in [Bra06] and the last one in [Ser07].

The computation of common subsumers is the inference service central to the bottom-up extension and to the customization of background ontologies. In this thesis we want to extend existing approaches for the computation of least common subsumers to more expressive DLs for use in ontology extension. A major difficulty of computing the lcs for ontology extension is the use of disjunction. Computing the lcs of the concepts $C_1, \ldots, C_n$ in a DL offering disjunction, results in the disjunction $C_1 \sqcup \cdots \sqcup C_n$ as their lcs. This result is correct, but useless for our application. In the above described extension approaches supporting users in modeling, the concept description returned by the lcs is shown to the user for inspection and editing. Now, in order to assess whether the proposed concept description captures a certain characteristic of the aspired concept, the modeler needs to comprehend which information is captured in the concept description and which is not. In addition to this, the modeler has to know where the information is stated in order to edit the concept description competently. To this end the concept description has to fulfill two requirements:

- It needs to be compact to be easily comprehensible, and
- it should explicitly state the information common to all input concepts.

The latter is surely not fulfilled by just enumerating the input concepts as done in the plain disjunction for the lcs. Many DLs in use for practical applications as well as OWL Lite and OWL DL allow for disjunction. In order to offer automated support for the extension of ontologies written in these ontology languages by the approaches sketched above, methods to compute 'meaningful' common subsumers in the presence of disjunction are required. In this thesis we propose and investigate two approaches for this.

## 1.3 Structure of the thesis

In this section we provide an overview of the results achieved in the course of this Ph.D. project on non-standard inferences. To some extent these results have already been published.

In this thesis we present two approaches on how to compute common subsumers for concept descriptions written in expressive DLs. We devise computation algorithms for the inferences that realize these two approaches, provide and describe their implementation and a first qualitative evaluation of it. In particular, the proposed approaches address the problem how to obtain meaningful concept descriptions for common subsumers in DLs that allow for concept disjunction. In particular, we propose

1. the approximation-based approach to compute common subsumers and

2. the computation of common subsumers w.r.t. a background ontology.

In preparation for this Section 2 introduces formal preliminaries on DLs and their basic inferences.

In Section 3 we give an overview on methods for the computation of the lcs. We discuss in depth the approach for the DL $\mathcal{ALE}$ by Baader, Küsters and Molitor presented in [BKM99], which is the basis for the results in the following sections. The lcs for more than two concept descriptions obtained by this method can grow exponentially in the size of the initial concepts. These concept descriptions cannot be represented in a more compact form even if one allows to introduce new concept definitions for sub-concept descriptions used in the lcs concept description in the TBox. This result has been already been published in [BT01a; BT02a]. In Section 3.3 we discuss the method by Küsters and Molitor to compute the lcs for $\mathcal{ALEN}$-concept descriptions [KM01b].

Besides the results on the compact representation of least common subsumers, we have also devised a method for computing the lcs for DLs with transitive roles in [BT03; BTK03a]. These results, however, will not be discussed in this thesis in detail.

- [BT01a]
  F. Baader and A.-Y. Turhan. TBoxes do not yield a compact representation of the least common subsumer. In C. Goble, R. Möller, and P.F. Patel-Schneider, editors, *Proc. of the 2001 Description Logic Workshop (DL 2001)*, number 49 in CEUR Workshop, 2001.

- [BT02a]
  F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In M. Jarke, J. Köhler, and G. Lakemeyer, editors, *Proc. of the 25th German Annual Conf. on Artificial Intelligence (KI'02)*, vol. 2479 of *LNAI*. Springer, 2002.

- [BT03]
  S. Brandt and A.-Y. Turhan. Computing least common subsumers for $\mathcal{FLE}^+$. In D. Calvanese, G. De Giacomo, and E. Franconi, editors, *Proc. of the 2003 International Workshop on Description Logics*, number 81 in CEUR Workshop, 2003.

- [BTK03]
  S. Brandt, A.-Y. Turhan, and R. Küsters. Extensions of non-standard inferences for Description Logics with transitive roles. In M. Vardi and A. Voronkov, editors, *Proc. of the 10th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'03)*, vol. 2850 of *LNCS*. Springer, 2003.

Then we devise methods for the two above mentioned approaches on how to compute meaningful common subsumers in the presence of disjunction.

### Approximation-based approach for the computation of common subsumers

In this approach the input concept descriptions are first translated into a less expressive DL that does not offer concept disjunction; then the lcs of the approximated concept descriptions is computed in that DL. To this end we introduce the non-standard inferences:

**Concept Approximation.** This inference service was initially investigated for the $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concept descriptions in [BKT02b] (and recently discussed in [Bra06]). In Section 4 we extend the method for computing approximation presented there to number restrictions and prove soundness and completeness of the computation method.

**Difference operator.** By the translation into a less expressive DL, information of the original concept is lost. To assess how much, or, more precisely, which parts of the concept descriptions were lost, we define the syntactic difference operator in Section 4.3. We devise a heuristic method to compute the syntactic difference of $\mathcal{ALE}$-concept from an $\mathcal{ALC}$-concept description. By 'subtracting' the approximated concept from the original concept, one can assess which information is not captured in the approximated concept.

The results on the approximation-based approach for the computation of common subsumers have been published in:

- [BKT02a]
  S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In I. Horrocks and S. Tessaris, editors, *Proc. of the 2002 Description Logic Workshop (DL 2002)*, number 53 in CEUR Workshop, 2002.

- [BKT02b]
  S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in Description Logics. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proc. of the 8th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-02)*, 2002.

- [BT01b]
  S. Brandt and A.-Y. Turhan. Using non-standard inferences in Description Logics — what does it buy me? In G. Görz, V. Haarslev, C. Lutz, and R. Möller, editors, *Proc. of the 2001 Applications of Description Logic Workshop (ADL 2001)*, number 44 in CEUR Workshop, 2001.

### Computation of Common Subsumers w.r.t. a Background Ontology

In Section 5 we study the new framework for computing least common subsumers, namely, w.r.t. a background knowledge base, which was introduced in [BST04a]. This framework proposes to customize a background ontology written in an expressive

DL by a user ontology written in a less expressive DL not offering disjunction. We investigate the customization of $\mathcal{ALC}$-background knowledge bases. In Section 5.2 we show existence of the lcs in the user DL $\mathcal{EL}$ and $\mathcal{ALE}$ w.r.t. unfoldable TBoxes. For general or cyclic TBoxes the lcs neither exists for $\mathcal{EL}$ nor for $\mathcal{ALE}$, which is shown in Section 5.3, see also [BST04b]. In some cases it seems to be useful to relax the notion of the lcs to avoid over-fitting or for efficiency reasons. To this end we describe practical approaches for computing *good* common subsumers, which may, however, not be the least ones. In Section 5.4 three of these approaches are discussed in detail.

Our results on the customization of background ontologies have been been published in:

- [BST04a]
  F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In V. Haarslev and R. Möller, editors, *Proc. of the 2004 Description Logic Workshop (DL 2004)*, number 104 in CEUR Workshop, 2004.

- [BST04b]
  F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J.J. Alferes and J.A. Leite, editors, *Proc. of the 9th European Conf. on Logics in A.I. (JELIA 2004)*, volume 3229 of *LNCS*, 2004

- [BST07]
  F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *Journal of Applied Logics*, 2007.

### Approaches put to practice

In Section 6 we describe the first implementation of the non-standard inferences introduced in the previous sections. Since the computation algorithms for lcs, concept approximation and good common subsumer are quite similar, we propose optimization techniques in Section 6.2 that are applicable to all of these inferences. Besides employing classical optimization techniques such as lazy unfolding and caching, we devise new techniques to reduce computation times. For instance, for the computation of concept approximation we have introduced a promising method to speed-up the computation in [BT02b] and extended in [TB07]. The idea is to perform concept approximation independently on each conjunct of a conjunction and then conjoin the result. This approach works only on a certain sub-class of $\mathcal{ALCN}$-concept descriptions. In Section 6.2.2 we give syntactic conditions to recognize the kind of concept descriptions and prove the correctness of these conditions. Results on earlier versions of these implementations have been published in [TM01; BT02a; BKT02b; BST07; TB07].

We complement our implementation of the approximation-based approach by implementations of a heuristic for minimal rewriting and the difference operator. These implementations were presented in [BKT02b; Tur05] and are discussed in Section 6.4 of this work.

Last in this section we describe our non-standard inference system SONIC [TK04a; TK04b; Tur05], which offers most of the non-standard inferences discussed here as a plug-in for the widely used ontology editor PROTÉGÉ. SONIC implements more inference services than discussed in detail in this work, which are shortly mentioned in Section 6.5. Besides this we report on efforts to provide an interface for other system developers to use non-standard inferences in their systems [TBK+06].

The publications on the implementation of non-standard inferences investigated in this thesis and on SONIC are:

- [BT02b]
  S. Brandt and A.-Y. Turhan. An approach for optimized approximation. In G. Görz, V. Haarslev, C. Lutz, and R. Möller, editors, *Proc. of the 2002 Applications of Description Logic Workshop (ADL 2002)*, nr. 63 in CEUR Workshop, 2002.

- [TB07]
  A.-Y. Turhan and Y. Bong. Speeding up approximation with nicer concepts. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, and A.-Y. Turhan, editors, *Proc. of the 2007 Description Logic Workshop (DL 2007)*, 2007.

- [TBK+06]
  A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithöner. DIG 2.0 – Towards a flexible interface for Description Logic reasoners. In B. Cuenca Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *In Proc. of the second international workshop OWL: Experiences and Directions*, 2006.

- [TK04a]
  A.-Y. Turhan and C. Kissig. SONIC — Non-standard inferences go OILED. In D. Basin and M. Rusinowitch, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-04)*, vol. 3097 of *LNCS*, 2004.

- [TK04b]
  A.-Y. Turhan and C. Kissig. SONIC—System description. In V. Haarslev and R. Möller, editors, *Proc. of the 2004 Description Logic Workshop (DL 2004)*, nr. 104 in CEUR Workshop, 2004.

- [TM01]
  A.-Y. Turhan and R. Molitor. Using lazy unfolding for the computation of least common subsumers. In C. Goble, R. Möller, and P.F. Patel-Schneider, editors, *Proc. of the 2001 Description Logic Workshop (DL 2001)*, nr. 49 in CEUR Workshop, 2001.

- [Tur05]
  A.-Y. Turhan. Pushing the SONIC border — SONIC 1.0. In R. Letz, editor, *Proc. of Fifth International Workshop on First-Order Theorem Proving*

*(FTP 2005)*. Technical Report University of Koblenz, 2005. http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-13-2005.pdf.

A qualitative evaluation of the proposed approaches and their implementations of the introduced inferences and w.r.t. quality of the obtained result is given in Section 7. This evaluation is based on knowledge bases from practical applications. We summarize and discuss the theoretical as well as the practical outcome of this work and point out aspects for future work in Section 8.

# Chapter 2

# Preliminaries

In this chapter we introduce the basic notions of DL systems in general and of the specific DLs used in this thesis and inferences formally, that so far have been only used in an intuitive way. We start by introducing the representation formalism and then define inferences typical for DL systems.

## 2.1 Concept constructors of Description Logics

DLs are based on the following sets of names: $N_C$ is the set of concept names, and $N_R$ is the set of role names. Complex concept descriptions are inductively defined starting from the set of concept names and using concept constructors. The concept constructors provided in DLs used in this thesis are shown in Table 2.1.

The semantics of concept descriptions are given in a set-theoretic way. It is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$. The domain $\Delta$ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta$. Each role name $r \in N_R$ is mapped to a binary relation $r^{\mathcal{I}} \subseteq \Delta \times \Delta$. Of a pair of individuals related via a role, we call the first one the *role predecessor* and the second one the *role successor*. Starting from an interpretation of primitive concepts ($C \in N_C$), the extension of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions is defined inductively, as shown in the second column of Table 2.1.

The top-concept $\top$ denotes the domain and thus is the most general concept. The bottom-concept $\bot$ denotes a contradictory concept. The *conjunction* and *disjunction* concept constructors are defined in the same way as in propositional logics. Negation for DLs is available in two forms as a concept constructor. The restricted form is called *primitive negation*, it allows negation only in front of concept names. The general form is called *full negation*, which can be used for arbitrary, complex concept descriptions. *Existential restrictions* ($\exists r.C$) enforce a role successor for the role $r$ that belongs to concept $C$. *Value restrictions* ($\forall r.D$) enforce that, if a role successor for the role $r$ exists, then this role successor must belong to concept $D$. An interval for the number or role successors w.r.t. a role can be specified by *number restrictions*. An upper bound can be introduced by an *at-most restriction* and a lower bound can be expressed by an *at-least restriction*. We illustrate the use of these concept constructors

| Constructor name | Syntax | Semantics |
|---|---|---|
| top-concept | $\top$ | $\Delta_{\mathcal{I}}$ |
| bottom-concept | $\bot$ | $\emptyset$ |
| concept name, $A \in N_C$ | $A$ | $A^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}}$ |
| conjunction | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| disjunction | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| primitive negation, $A \in N_C$ | $\neg A$ | $\Delta_{\mathcal{I}} \setminus A^{\mathcal{I}}$ |
| full negation | $\neg C$ | $\Delta_{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| existential restrictions | $\exists r.C$ | $\{x \in \Delta_{\mathcal{I}} \mid \exists y\!:\!(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| value restrictions | $\forall r.C$ | $\{x \in \Delta_{\mathcal{I}} \mid \forall y\!:\!(x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |
| number restrictions, $n \in \mathbb{N}$ | | |
| at-least restrictions | $(\geq n\ r)$ | $\{x \in \Delta_{\mathcal{I}} \mid \#\{y : (x,y) \in r^{\mathcal{I}}\} \geq n\}$ |
| at-most restrictions | $(\leq n\ r)$ | $\{x \in \Delta_{\mathcal{I}} \mid \#\{y : (x,y) \in r^{\mathcal{I}}\} \leq n\}$ |

Table 2.1: Syntax and semantics of concept descriptions.

by a toy example from the domain of air-planes.

**Example 1 (Concept constructors).** *We can characterize a plane by specifying its number of seats and passenger classes in different configurations as follows:*

$$\begin{aligned}&\mathsf{Plane} \sqcap (\geq 2\ \mathsf{has\text{-}config}) \sqcap \\ &\forall\, \mathsf{has\text{-}config}.(\mathsf{Passenger\text{-}Config} \sqcap (\geq 261\ \mathsf{has\text{-}seats})) \sqcap \\ &\big(\exists \mathsf{has\text{-}config}.((\leq 419\ \mathsf{has\text{-}seats}) \sqcap (\leq 2\ \mathsf{has\text{-}classes})) \sqcup \\ &\ \exists \mathsf{has\text{-}config}.((\leq 380\ \mathsf{has\text{-}seats}) \sqcap (\leq 3\ \mathsf{has\text{-}classes}))\big).\end{aligned}$$

*This concept description specifies planes that have only passenger configurations with at least 261 seats and that have a configuration with at most 419 seats in at least 2 passenger classes or it has a configuration with at most 380 seats in at least 3 passenger classes.*

Besides concept constructors most expressive DLs provide means to declare properties of roles directly. One can declare a role to be

- a *transitive role*, which is interpreted as a transitive relation.
- the *inverse role* of another role *inverse*$(R_1, R_2)$, which are interpreted as $R_2^{\mathcal{I}} = \{(a,b) \mid (b,a) \in R_1^{\mathcal{I}}\}$.
- a *super-role* of another one. Role inclusion axioms $R \sqsubseteq S$ enforce that every pair $(a,b) \in R^{\mathcal{I}}$ is also $(a,b) \in S^{\mathcal{I}}$. The set of these kind of statements form the *role hierarchy*.
- a *feature*, i.e., a functional role. The relations of this kind are interpreted as functions between elements of the domain $\Delta$.

| Constructor name | $\mathcal{EL}$ | $\mathcal{FLE}$ | $\mathcal{ALE}$ | $\mathcal{ALN}$ | $\mathcal{ALEN}$ | $\mathcal{ALC}$ | $\mathcal{ALCN}$ |
|---|---|---|---|---|---|---|---|
| top-concept | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| bottom-concept | | | ✗ | ✗ | ✗ | ✗ | ✗ |
| concept name | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| conjunction | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| disjunction | | | | | | ✗ | ✗ |
| negation | | | prim. | prim. | prim. | full | full |
| existential restrictions | ✗ | ✗ | ✗ | | ✗ | ✗ | ✗ |
| value restrictions | | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| at-least restrictions | | | | ✗ | ✗ | | ✗ |
| at-most restrictions | | | | ✗ | ✗ | | ✗ |

Table 2.2: Concept constructors for $\mathcal{ALCN}$ and sub-languages.

Each DL is characterized by the set of concept and possibly role constructors that the DL offers for building concept descriptions or characterizing roles. In this thesis we will use the DLs shown in Table 2.2, which do not allow to specify information on roles. For a wider overview on concept and role constructors for DLs please refer to [BN03; CG03]. If a concept description uses only concept constructors from a DL $\mathcal{L}$ it is called a $\mathcal{L}$-concept description, e.g., the concept description from Example 1 is an $\mathcal{ALCN}$-concept description.

## 2.2 TBox and ABox formalisms

As already mentioned in Section 1, DL knowledge bases are divided into two parts: TBox and ABox. The former represents the *terminological knowledge*, i.e., it formalizes the basic categories of the application domain. The latter represents the *assertional knowledge*, i.e., formalizes the notions of individual entities from this domain. We define these two notions now formally.

### 2.2.1 TBoxes

The TBox captures the terminological knowledge of the application domain. The form of TBoxes early DL systems were able to handle is rather simple in contrast to statements supported by current DL systems. The order in which we define TBox statements in this subsection reflects this historical development. In the early DL systems, TBoxes basically introduced a concept name as an abbreviation for a concept description by a concept definition.

**Definition 2 ((Primitive) concept definition).** Let $A$ and $B$ be concept names and $C$ and $D$ be concept descriptions, then

- the expression $A \sqsubseteq C$ is a *primitive concept definition* and $C$ is called a *primitively defined concept*.

- the expression $B \equiv D$ is a *concept definition* and $D$ is called a *defined concept*.

The semantics of (primitive) concept definitions are given by the interpretation function: $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and $B^{\mathcal{I}} = D^{\mathcal{I}}$. ◇

A primitive concept definition specifies only the necessary conditions for an instance of that concept, while a concept definition gives necessary *and* sufficient conditions for an instance of that concept. Please note that for DLs with primitive negation as concept constructors, only the concepts without a (primitive) definition may be used in their negated form in the TBox.[7]

To avoid confusion, please note that by *primitive* concepts we understand concepts that do not have a definition and by *primitively defined* concepts we understand concepts with a primitive definition in the terminology. If in a concept definition $C \equiv D$ the concept description $D$ refers directly or indirectly to the concept name $C$, we call such a concept $C$ a *cyclic concept* and $C \equiv D$ a *cyclic concept definition*. Based on this, we define the notion of a TBox.

**Definition 3 (TBox).** Let $A, B$ be concept names and let $C, D$ be concept descriptions. A finite set of possibly primitive concept definitions $\mathcal{T}$ is called a *TBox*, if

- all definitions are acyclic and
- for each concept name $A$ there is at most one concept definition in $\mathcal{T}$ with $A$ on the left-hand side.

An interpretation is a *model* of a TBox, if for all $A \sqsubseteq C \in \mathcal{T}$ and $B \equiv D \in \mathcal{T}$ it holds that $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and $B^{\mathcal{I}} = D^{\mathcal{I}}$. ◇

A set of concept descriptions that contains one or more cyclic concepts is called a *cyclic TBox*, while a TBox that contains no such concept is an *acyclic TBox*. A TBox that is written in a DL $\mathcal{L}$ is called a *$\mathcal{L}$-TBox*. In TBoxes written in a DL that offers only primitive negation, negation may only appear in front of primitive concepts.

**Example 4 (TBox).** *We return to our domain of planes and state that passenger configurations are not cargo configurations. We assign the concept description from Example1 a concept name.*

$$\begin{aligned}\mathcal{T}_{Airbus} = \{\mathsf{Cargo\text{-}Config} &\equiv \neg\mathsf{Passenger\text{-}Config},\\ \mathsf{Airbus\text{-}340} &\equiv \mathsf{Plane} \sqcap (\geq 2\ \mathsf{has\text{-}config}) \sqcap\\ &\quad \forall\, \mathsf{has\text{-}config}.(\mathsf{Passenger\text{-}Config} \sqcap (\geq 261\ \mathsf{has\text{-}seats})) \sqcap\\ &\quad \big(\exists \mathsf{has\text{-}config}.((\leq 419\ \mathsf{has\text{-}seats}) \sqcap (\leq 2\ \mathsf{has\text{-}classes})) \sqcup\\ &\quad\ \exists \mathsf{has\text{-}config}.((\leq 380\ \mathsf{has\text{-}seats}) \sqcap (\leq 3\ \mathsf{has\text{-}classes}))\big)\ \}\end{aligned}$$

*In this TBox the concepts* Cargo-Config *and* Passenger-Config *are disjoint concepts. The concept* Airbus-340 *is defined in terms of the concept* Passenger-Config*. The concept definition of* Airbus-340 *states that all configurations are* Passenger-Config *with*

[7] Otherwise the negation is no longer primitive, but refers to arbitrary concepts.

*261 seats or more. Furthermore the definition describes two configurations w.r.t. the maximal number of passenger classes and seats that these configurations offer.*

*Although in this TBox negation appears only in front of primitive concept names, this TBox is an $\mathcal{ALCN}$-TBox, since disjunction is used.*

The TBoxes according to Definition 3 are sometimes called *unfoldable TBoxes*. Intuitively, unfoldable TBoxes are TBoxes in which each occurrence of a defined concept can be replaced by the left-hand side of its concept definition, without losing information or without termination problems due to cycles.

Early DL systems were able to handle only unfoldable terminologies. It turned out that for some applications this form of TBoxes is too restrictive and a more expressive form of TBox statements was introduced. This kind of statements allows to establish sub-concept super-concept relationships between arbitrary concept descriptions, i.e., without having to introduce concept names for these concept descriptions. For instance, it might be convenient to state in our planes TBox directly, that planes with more than 10 seats are required to have at least two emergency exits in the following way: $\mathsf{Plane} \sqcap (\geq 10\ \mathsf{has\text{-}seats}) \sqsubseteq (\geq 2\ \mathsf{has\text{-}Emergency\text{-}Exits})$. These kind of statements are called general concept inclusion axioms.

**Definition 5 (GCI, general TBox).** Let $C_1$, $C_2$, $D_1$ and $D_2$ be concept descriptions, then

$$C_1 \sqsubseteq C_2$$

is a *general concept inclusion axiom* (GCI). The semantics of GCIs is given by the interpretation function. A GCI $C_1 \sqsubseteq C_2$ is satisfied for a TBox $\mathcal{T}$, iff $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{T}$.

A TBox $\mathcal{T}$ is a *general TBox* if it contains GCIs. An interpretation is a *model* of a general TBox, if for all $C_1 \sqsubseteq C_2 \in \mathcal{T}$ and $D_1 \equiv D_2 \in \mathcal{T}$, it holds that $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ and $D_1^{\mathcal{I}} = D_2^{\mathcal{I}}$. ◇

It is easy to see that GCIs are a more general form of primitive concept definitions and defined concepts as introduced in Definition 2 can be simulated by a pair of GCIs in the following way:

$$C_1 \sqsubseteq C_2 \ \ \textit{and} \ \ C_2 \sqsubseteq C_1.$$

All recent DL systems support inference services for general TBoxes. In this thesis we consider mainly unfoldable TBoxes in which the concept constructors from Table 2.1 are used. Next, ABoxes are introduced formally, they have so far only been used in an intuitive way.

### 2.2.2 ABoxes

The knowledge about individual entities from the application domain can be expressed by so-called *ABox assertions*. There are two kinds of ABox assertions used for DL systems—one kind expresses that an individual belongs to a concept and the other one specifies that two individuals are related via a role. The set $N_I$ is the set of all individual names.

**Definition 6 (ABox, ABox assertion).** Let $C$ be an arbitrary concept description, $r \in NR$ be a role name and $i, j$ ($\{i, j\} \subseteq N_I$) be two individual names, then

- $C(i)$ is called an *concept assertion* and
- $r(i, j)$ is called a *role assertion*.

An ABox $\mathcal{A}$ is a set of concept assertions and role assertions. $\diamond$

The term *ABox assertions* comprises both, concept assertions and role assertions. If all concepts in an ABox $\mathcal{A}$ are from a Description Logic $\mathcal{L}$, then we call $\mathcal{A}$ a $\mathcal{L}$-ABox.

In order to capture ABoxes the interpretation function is now extended to individual names, which are mapped to elements of the domain $\Delta$. In DL-systems it is typically assumed that distinct names denote distinct objects. This assumption is called *unique name assumption* (UNA). The mapping of individual names has to respect this assumption, thus if $a \neq b$ then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

**Definition 7 (Semantics of ABoxes, semantics of assertions).** Let $C$ be an arbitrary concept, $r$ be a role name and $i, j$ be two individuals, then an interpretation $\mathcal{I}$ satisfies

- the concept assertion $C(i)$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$ and
- the role assertion $r(i, j)$ iff $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in r^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ satisfies an ABox $\mathcal{A}$ iff $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$. In this case $\mathcal{I}$ is a *model of* $\mathcal{A}$ . $\diamond$

Based on these formal semantics of ABox and TBox statements, a variety of inferences have been defined. This thesis is dedicated to non-standard inferences, which are a relatively new group of inferences. These inferences are often defined based on the older, well-established 'standard inferences' such as satisfiability or subsumption, which we introduce in the next section.

## 2.3 Inferences of Description Logic systems

DL systems offer different kinds of inferences to make knowledge that is implicitly captured in the knowledge base explicit. A collection of inferences that was investigated already in the early years of DL research and is nowadays implemented in most DL reasoning systems are the standard inferences.

### 2.3.1 Standard inferences of Description Logic systems

When writing a concept definition to add it to a DL knowledge base, it is crucial to know whether the specified concept description contains a contradiction (w.r.t. the knowledge base) or whether it could be fulfilled by any individual. This leads to the formal notion of consistency.

**Definition 8 (Concept consistency, TBox consistency).** Let $C$ be a concept description and $\mathcal{T}$ be a TBox. The concept description $C$ is *consistent* iff it has a model, i.e., iff there exists an interpretation $\mathcal{I}$ where $C^{\mathcal{I}} \neq \emptyset$. In this case $\mathcal{I}$ is a model of $C$. A TBox $\mathcal{T}$ is *consistent* iff every concept in $\mathcal{T}$ is consistent. $\diamond$

If a concept or TBox is not consistent, it is called *inconsistent*. The actual inference service that determines whether a concept (/TBox) is satisfiable is called *concept (/TBox) consistency*. Alternatively, we say that a concept (/TBox) is *satisfiable*.

Another terminological inference task is to determine whether one concept description is more general than another, i.e., whether one concept description $C$ is implied by another concept description $D$. This is the case, if every individual that is an instance of $C$ also is an instance of $D$. The following definition formalizes this notion of subsumption.

**Definition 9 (Concept subsumption, equivalence of concepts).** Let $C, D$ be two concept descriptions and $\mathcal{T}$ be a (possibly empty) TBox. The concept description $C$ is *subsumed w.r.t.* $\mathcal{T}$ by the concept description $D$ ($C \sqsubseteq_{\mathcal{T}} D$), iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{T}$. Two concept descriptions $C, D$ are *equivalent w.r.t.* $\mathcal{T}$ ($C \equiv_{\mathcal{T}} D$), iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for every model $\mathcal{I}$ of $\mathcal{T}$. $\diamond$

A test for the equivalence of concept descriptions can be reduced to two subsumption tests, since the following holds: ($C \equiv_{\mathcal{T}} D$), iff $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$.

One of the most important traditional inference services provided by DL systems based on subsumption is the computation of the *subsumption hierarchy* of all named concepts in the TBox or the *taxonomy* of the TBox. This inference service is called *classification*.

Besides reasoning services for concept descriptions or TBoxes, there are also well-investigated inferences that reason over the information captured in the ABox. For an overview of all ABox inference services see [BN03]. For our discussion in this thesis we need to extend the notion of consistency to ABoxes.

**Definition 10 (Instance of, ABox consistency).** Let $C$ be an arbitrary concept description and $i$ ($i \in N_I$) be an individual name and $\mathcal{A}$ an ABox. The individual $i$ is an *instance of* $C$ w.r.t. an ABox $\mathcal{A}$, iff $a^{\mathcal{I}} \in_{\mathcal{A}} C^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ of $\mathcal{A}$. The test whether an individual is an instance of a concept (w.r.t. an ABox) is called *instance checking*. An ABox $\mathcal{A}$ is consistent, iff every individual $i$ in $\mathcal{A}$ is consistent w.r.t. $\mathcal{A}$. $\diamond$

We saw earlier that the equivalence of concepts can be reduced to subsumption. Similarly, one can reduce the other reasoning services to subsumption provided that full negation is present in the underlying DL. For TBoxes written in DLs with full negation in their set of concept operators, subsumption can be reduced to consistency, see [Smo88]. One can reduce

- *subsumption to (in-)consistency*
  $C \sqsubseteq_{\mathcal{T}} D$ iff the concept $C \sqcap \neg D$ is consistent w.r.t. $\mathcal{T}$.

- *instance checking to (in-)consistency*
  An individual $i$ is an instance of a concept $C$ w.r.t. a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, iff $\mathcal{A}' := \mathcal{A} \cup \{\neg C(i)\}$ is inconsistent.

Due to these reductions it suffices for DLs that offer full negation in principle to investigate and implement a method for only one of these inferences to obtain methods for the other ones. A natural questions is what the complexity of computing an inference is.

### 2.3.2 Complexity of terminological reasoning

The methods for the computation of non-standard inferences (NSIs) in general and of common subsumers in particular rely on the methods for satisfiability and subsumption. We recall complexity results on satisfiability and subsumption for the DLs introduced in Table 2.2. For a broader overview on the complexity of terminological reasoning in more expressive DLs the reader is referred to [Don03].

The worst case complexities for reasoning in DLs are given w.r.t. the size of the input concept descriptions, TBox or ABox, which we define next for the DL $\mathcal{ALCN}$ (and its sub-languages).

**Definition 11 ($\mathcal{ALCN}$-concept size, TBox size, ABox size).** Let $C$ and $D$ be $\mathcal{ALCN}$-concept descriptions. Then the size of a concept description $C$ (denoted $|C|$) is defined as:

$$\begin{aligned} |\top| &:= 0 & |\bot| &:= 0 \\ A \in N_C, |A| &:= 1 & |\neg C| &:= |C| \\ |C \sqcap D| &:= |C| + |D| & |C \sqcup D| &:= |C| + |D| \\ |\forall r.C| &:= |C| + 1 & |\exists r.C| &:= |C| + 1 \\ |(\leq n\ r)| &:= 0 & |(\geq n\ r)| &:= 0. \end{aligned}$$

The *size of a (primitive) concept definition or GCI* is the sum of the concept descriptions from the left- and right-hand side. The *size of a TBox* $\mathcal{T}$ ($|\mathcal{T}|$) is defined as the sum of the sizes of all TBox statements in $\mathcal{T}$,

The *size of an ABox assertion* is 1 for role assertions and for concept assertions $a : C$ the size is $|C|$. The *size of an ABox* $\mathcal{A}$ ($|\mathcal{A}|$) is then the sum of all ABox assertions in $\mathcal{A}$. ◇

DLs that do not offer all propositional operators to describe concepts are called *sub-Boolean* DLs. Besides $\mathcal{ALC}$ and $\mathcal{ALCN}$ all DLs characterized in Table 2.2 are sub-Boolean. For this kind of DLs, subsumption cannot be reduced to satisfiability as described above. Thus different methods for satisfiability and subsumption have been devised.

In DLs that cannot express contradictions, such as $\mathcal{EL}$ or $\mathcal{FLE}$, satisfiability of concept descriptions is trivial. In $\mathcal{ALN}$, where inconsistencies can be expressed, satisfiability of concept descriptions can be computed in polynomial time [DLNN97].[8] In

[8] We assume basic familiarity with basic notions of complexity classes as, for instance, introduced in [Pap94].

| | Satisfiability | | Subsumption | |
|---|---|---|---|---|
| $\mathcal{EL}$ | trivial | | P | [BKM99] |
| $\mathcal{ALN}$ | P | [DLNN97] | P | [DLNN97] |
| $\mathcal{FLE}$ | trivial | | NP-complete | [DLN+92] |
| $\mathcal{ALE}$ | NP-complete | [DLN+92] | NP-complete | [DLN+92] |
| $\mathcal{ALEN}$ | PSPACE | [DLNN97] | PSPACE-complete | [Hem01] |

Table 2.3: Complexity of concept satisfiability and subsumption in sub-Boolean DLs.

the presence of existential and value restrictions, satisfiability for $\mathcal{ALE}$-concept descriptions is NP-complete [DLN+92], while for $\mathcal{ALEN}$ satisfiability is in PSPACE [DLNN97]. These results are summed up in Table 2.3 together with the complexity results for checking subsumption and the resp. reference to the literature where the result was proven. The complexity of testing subsumption between concept descriptions is not much harder in terms of complexity classes. However, things change in case subsumption is computed w.r.t. TBoxes. Even for a DL that just offers conjunction and value restrictions, checking subsumption w.r.t. a TBox is CO-NP-complete [Neb90] due to unfolding of the concepts w.r.t. the TBox.

For the DLs that are closed under negation, the complexity results for subsumption and satisfiability coincide. Subsumption of $\mathcal{ALC}$- and $\mathcal{ALCN}$-concept descriptions is PSPACE-complete [SS88]. Besides the set of concept constructors provided by a DL, also the kind of concept statements allowed in the TBox contribute to the complexity of reasoning—it is often significantly higher when GCIs are allowed in the TBox. So, for $\mathcal{ALC}$ subsumption w.r.t. an unfoldable TBox is still PSPACE-complete [Lut99], while subsumption w.r.t. general TBoxes is in EXPTIME [DM00].

However, $\mathcal{SHIN}$ and $\mathcal{SHIF}$—the DLs that are underlying OWL DL and OWL Lite—do not offer more concept constructors than $\mathcal{ALCN}$, but allow to specify information on roles. In particular $\mathcal{SHIN}$ and $\mathcal{SHIF}$ offer all role operators introduced earlier on page 24. Usually, reasoning for these very expressive DLs is only investigated w.r.t. general TBoxes. In case of $\mathcal{SHIN}$ and $\mathcal{SHIF}$ testing subsumption w.r.t. general TBoxes is EXPTIME-complete [Tob01].

## 2.4 Basic non-standard inferences

The collection of non-standard inferences for DLs has been extended in the last decade; for an overview on these inferences see [BK05]. Section 1.2.2 introduced basic non-standard inferences in an intuitive way, now we formally define those NSIs that are used in the bottom-up extension of knowledge bases.

### 2.4.1 Least common subsumer

Given a collection of concept descriptions in a DL $\mathcal{L}$, the least common subsumer (lcs) returns the concept description in $\mathcal{L}$ that subsumes all elements from the collection

and is the least concept description w.r.t. subsumption with this property.

**Definition 12 (Least common subsumer).** Let $\mathcal{T}$ be a TBox and $D$, $C_0, \ldots, C_n$ concepts in $\mathcal{L}$, then $D$ is the *least common subsumer (lcs)* of $C_0, \ldots, C_n$ w.r.t. $\mathcal{T}$ (written $\mathsf{lcs}_{\mathcal{L}}(C_0, \ldots, C_n)$) iff

1. $C_i \sqsubseteq_{\mathcal{T}} D$ for all $0 \leq i \leq n$, and
2. for all $\mathcal{L}$-concepts $C'$, $C_i \sqsubseteq_{\mathcal{T}} C'$ for all $0 \leq i \leq n$ implies $D \sqsubseteq_{\mathcal{T}} C'$.

$\diamond$

If the DL $\mathcal{L}$ we refer to is evident from the context, we write $\mathsf{lcs}$ instead of $\mathsf{lcs}_{\mathcal{L}}$ throughout the thesis.

In general (i.e., for an arbitrary DL $\mathcal{L}$), a given collection of $n$ concept descriptions need not have a least common subsumer. For example in a DL $\mathcal{L}$ that does not offer conjunction, there may be several concept descriptions in $\mathcal{L}$ that are incomparable (w.r.t. subsumption) and that subsume the input concept descriptions. Thus Condition 2 of the definition above is not fulfilled for some input concept descriptions of the DL $\mathcal{L}$.

Please note that Definition 12 implies that the lcs of an empty set of concept descriptions is $\bot$, if $\bot$ is provided by the DL. It follows directly from the definition of conjunction and lcs that the lcs of a set of concept descriptions in a DL $\mathcal{L}$ offering conjunction is unique up to equivalence. Thus it is justified to speak about *the lcs* in these cases. Furthermore the lcs is an associative operation, consequently we can obtain the lcs of $C_1, \ldots, C_n$ by computing: $\mathsf{lcs}(C_1, \mathsf{lcs}(C_2, \ldots \mathsf{lcs}(C_{n-1}, C_n) \ldots)$ and it suffices to devise a binary operation for the lcs.

The lcs inference is the basis for our investigation of methods for computing 'meaningful' common subsumers for concept descriptions written in DLs offering disjunction. We discuss existing approaches for computing least common subsumers in Chapter 3 and continue here with the other NSIs employed in the bottom-up extension of knowledge bases.

### 2.4.2 Most specific concept

Before the lcs is applied during bottom-up extension, the individuals from the ABox have to be generalized into concept descriptions. This is achieved by computing the most specific concept of each selected individual.

**Definition 13 (Most specific concept).** Let $\mathcal{A}$ be a $\mathcal{L}$-ABox, $i$ an individual in $\mathcal{A}$ and $C$ a $\mathcal{L}$-concept, then $C$ is the *most specific concept* (msc) of $i$ w.r.t. $\mathcal{A}$ (written $\mathsf{msc}_{\mathcal{A}}(i)$) iff

1. $i \in_{\mathcal{A}} C$, and
2. for all $\mathcal{L}$-concepts $D$, $i \in_{\mathcal{A}} D$ implies $C \sqsubseteq D$.

$\diamond$

The msc need not always exist. For the DLs $\mathcal{L}$ that we consider here, cyclic relations between individuals in the ABox cannot be expressed in $\mathcal{L}$-concept descriptions. For example, the cyclic ABox $\mathcal{A} = \{i{:}C,\ (i,i){:}r\}$ cannot be captured in an $\mathcal{ALE}$-concept description that is the *most* specific concept. For each concept description that captures $n$ loops as in, say $C^1 \sqcap \exists R.(C^2 \sqcap \ldots \exists R.C^n)$, there exists a more specific one that captures more loops and thus is more specific.

This has been remedied to a certain extent for the case of computing the msc in $\mathcal{ALE}$ by proposing *$k$-approximations* in [KM01a]. A $k$-approximation of an individual is the most specific concept with the maximal nesting depth of the quantifiers limited by a fixed constant $k$. In [KM01a; KM02] a method to compute these $k$-approximations is investigated.

A different approach was taken in [BK98] where the msc is captured by cyclic TBoxes with greatest fixed-point semantics. For this kind of TBox, the computation methods for the msc w.r.t. cyclic ABoxes have been devised for $\mathcal{ALN}$ in [BK98]. The msc for $\mathcal{ALN}$ can grow exponentially in the size of the ABox and can be computed in double-exponential time.

### 2.4.3 Minimal rewriting

The concept descriptions obtained by the msc or by the lcs can grow too large to be comprehensible by a human reader. If such a concept description is computed w.r.t. an unfoldable TBox, the concept description can often be 'compressed' by replacing subconcept descriptions by concept names from the TBox.[9] To this end *minimal rewriting* for DLs has been proposed in [BK00] as an instance of a more general framework for rewriting for Description Logics, which is defined as follows.

**Definition 14 (Rewriting).** Let $N_R$ be a set of role names and $N_P$ a set of primitive names, and let $\mathcal{L}_s$, $\mathcal{L}_d$, and $\mathcal{L}_t$ be three DLs (the source-, destination-, and TBox-DL, respectively). A *rewriting problem* is given by

- an $\mathcal{L}_t$-TBox $\mathcal{T}$ containing only role names from $N_R$ and primitive names from $N_P$; the set of defined names occurring in $\mathcal{T}$ is denoted by $N_D$;
- an $\mathcal{L}_s$-concept description $C$ using only the names from $N_R$ and $N_P$;
- a binary relation $\rho \subseteq \mathcal{L}_s \times \mathcal{L}_d$ between $\mathcal{L}_s$- and $\mathcal{L}_d$-concept descriptions.

An *$\mathcal{L}_d$-rewriting of $C$ using $\mathcal{T}$* is an $\mathcal{L}_d$-concept description $E$ built using names from $N_R$ and $N_P \cup N_D$ such that $C \rho E$.

Given an appropriate ordering $\preceq$ on $\mathcal{L}_d$-concepts, a rewriting $E$ is called *$\preceq$-minimal* iff there does not exist a rewriting $E'$ such that $E' \prec E$. ◇

The minimal rewriting problem is an instance of this rewriting framework, where $\mathcal{L}_s = \mathcal{L}_d = \mathcal{L}_t$, the binary relation $\rho$ corresponds to $\equiv_{\mathcal{T}}$ and the ordering $\preceq$ is w.r.t. concept size.

[9] This is an inverse process to unfolding.

**Definition 15 (Minimal rewriting).** Let $\mathcal{T}$ be a $\mathcal{L}$-TBox, $C, C', D$ be $\mathcal{L}$-concept descriptions. $C'$ is a *minimal rewriting* of $C$ w.r.t. $\mathcal{T}$, iff

1. $C'$ uses only concept and role names appearing in $\mathcal{T}$,
2. $C \equiv_{\mathcal{T}} C'$, and
3. for all $D$ where $D$ uses only concept and role names appearing in $\mathcal{T}$ and $C \equiv_{\mathcal{T}} D$, $|C'| \leq |D|$ holds.

◇

Regarding minimal rewriting the decision problem asks whether given a concept description and a TBox, a minimal rewriting of size $\kappa$ exists. The decision problem has been investigated for the DLs $\mathcal{ALN}$, $\mathcal{ALE}$ and $\mathcal{ALC}$. For the first two DLs deciding whether a minimal rewriting exists is NP-hard, whereas for $\mathcal{ALC}$ it is PSPACE-hard [BKM00].

For our task to represent the concept descriptions obtained by other NSIs in a compact way, the computation problem is the more interesting. More precisely, we need a method to obtain *one* minimal rewriting—instead of *all* possible minimal rewritings. In [BKM00] such a method has been devised for $\mathcal{ALE}$. It turned out that the algorithm presented in [BKM00] for computing a minimal rewriting in $\mathcal{ALE}$ is in PSPACE. The authors proposed a heuristic algorithm that computes small, but not always minimal rewritings. These rewritings can be computed in polynomial time (given an oracle for subsumption).

### 2.4.4 Concept matching

The inference service of concept matching was proposed in [McG96] to prune concept descriptions that are to be displayed to users of DL systems. Intuitively, concept matching can be used to search for structurally similar concepts in the TBox. So, in this application of pruning, concept matching was used to identify parts of the concept descriptions that were of interest to the user and thus should not be omitted when the concept is displayed.

In order to define a matching problem, we have to recall concept patterns first, which extend concept descriptions by variables. Let $N_X$ be a finite set of *concept variables* disjoint to $N_C \cup N_R$. *$\mathcal{L}$-concept patterns* are $\mathcal{L}$-concepts for which in addition concept variables can be used in the place of concept names—except for the fact that the primitive negation ($\neg$) may not occur in front of variables. A *substitution* $\sigma$ is a mapping from $N_X$ into the set of $\mathcal{L}$-concepts. It is extended to concept patterns $P$ by replacing every occurrence of $X \in N_X$ in $P$ by $\sigma(X)$. Thus, $\sigma(P)$ again is an $\mathcal{L}$-concept. With these preliminaries we can define matching problems as follows:

**Definition 16 (Matching problem, matcher).** An *$\mathcal{L}$-matching problem* is of the form $C \equiv^? P$, where C is an $\mathcal{L}$-concept and $P$ an $\mathcal{L}$-concept pattern. A substitution $\sigma$ is a *matcher* for $C \equiv^? P$ iff $C \equiv \sigma(P)$, i.e., $\sigma$ replaces the variables in $P$ by concepts in such a way that equivalence holds. ◇

The matching problem modulo subsumption can be reduced to matching modulo equivalence, since $C \sqsubseteq \sigma(D)$ iff $C \sqcap \sigma(D) \equiv C$. The decision problem whether a given matching problem has a matcher has been investigated in [BK00] for $\mathcal{EL}$ and $\mathcal{ALE}$. It turned out that deciding the solvability of a matching problem modulo subsumption (equivalence) is in P (NP-complete) for $\mathcal{EL}$, while for $\mathcal{ALE}$ both kinds of solvability problems are NP-complete.

In our application we are more interested in obtaining the actual matchers. A solvable matching problem may have infinitely many solutions. In the case of matching modulo subsumption one would like to obtain 'interesting' matchers, i.e. matchers that are close to $C$. This can be ensured to some extend by searching for *minimal matchers*, which are substitutions $\sigma$ for which no other substitution $\delta$ exists with $C \sqsubseteq \delta(D) \sqsubset \sigma(D)$. Methods for computing matchers in $\mathcal{ALE}$ and $\mathcal{ALN}$ (and their sub-languages) have been investigated in [BK00; Küs01].

The set of computed matchers can be restricted further by *side conditions*, which allow to specify a subsumer for each variable in the concept pattern. Matching under side conditions has been investigated for $\mathcal{ALN}$ [BRKM99; BBK01]. It has been shown in [BRKM99] that deciding the solvability of matching problems under strict subsumption conditions is NP-hard in $\mathcal{ALN}$ and its sub-languages. Moreover, concept matching in $\mathcal{ALN}$ under subsumption conditions is in P, even if the relevant system of subsumption conditions are cyclic [BBK01].

A novel approach was taken in [Bra06] where concept matching is computed w.r.t. *hybrid TBoxes*. This kind of TBoxes are split in a general and a (possibly) cyclic part. While the general part is interpreted w.r.t. greatest fixed-point semantics, the cyclic part is interpreted w.r.t. descriptive semantics, i.e. the 'usual DL semantics' as introduced in Definition 3.

In [Küs01; Bra06] the authors describe applications of concept matching for the design and maintenance of DL knowledge bases beyond the approach 'extension by modification' mentioned in the Introduction. However, the above mentioned computation algorithms for concept matching use the lcs as a sub-procedure. In order to obtain efficient implementations for concept matching, one needs efficient implementations for the lcs first.

# Chapter 3

# Least common subsumer

Our main motivation for investigating and implementing the lcs is the population of ontologies. In [BKM99] the *Bottom-up construction of knowledge bases* was introduced. This bottom-up extension provides means to generate concept descriptions in an example-driven way. In order to introduce a new concept, the knowledge engineer selects individuals from the ABox that are somewhat similar and which capture the characteristics of the category the knowledge engineer has in mind. The DL system automatically proposes a concept description for the intended concept generated from the selected ABox individuals. Such a concept description can be generated by employing two non-standard inferences: First, each of the selected individuals is generalized into a concept description by computing its msc. Second, these concept descriptions are then generalized into one single concept description by the lcs operation. The obtained concept description—possibly after computing its minimal rewriting—is then displayed to the knowledge engineer for inspection and for editing.

As mentioned in Chapter 1, a modified version of this bottom-up approach is to extend concept hierarchies that are not well balanced, because they are often unpractical for browsing and finding concepts in the terminology for re-use. In addition, such a shape of the taxonomy makes it more difficult for a modeler to comprehend the structure of a terminology. Thus it is desirable to introduce new concepts that yield new intermediate concepts in the taxonomy of the TBox. Ideally one would apply the lcs to sibling concepts from the concept hierarchy to extend the taxonomy with a concept generalizing a subset of the sibling concepts.

In this chapter we examine related work on applications and theoretical results for the lcs. We examine the computation methods for the lcs in the DLs $\mathcal{ALE}$ and $\mathcal{ALEN}$ in detail. These methods from [BKM99; KM01b] are based on the structural characterizations of subsumption, which are in turn the basis for our extension of the computation of 'meaningful' common subsumers to disjunction. We also present our result on the compact representation of common subsumers in $\mathcal{ALE}$, which carry over to the case of $\mathcal{ALEN}$. The chapter ends with a discussion of the drawbacks of the proposed methods with respect to the support of ontology extension.

## 3.1 Related work on the lcs

The inference least common subsumer was first introduced by Cohen, Hirsh and Borgida [CBH92] for DLs to provide learning of concept descriptions for the early DL system Classic [PMB+91; MH03]. Since then this inference service was investigated for a small range of DLs and employed in a number of applications.

### 3.1.1 Applications of the lcs

The applications for the lcs can be roughly grouped into three categories: the support for building ontologies, the similarity-based indexing or retrieval of information and the lcs is employed to support other reasoning tasks.

**Building Ontologies.** To learn concept descriptions either from sets of individuals or from sets of concepts was in fact the initial motivation to define and investigate the lcs inference. In the domain of automated learning, the task 'learning from examples' is to find the most specific concept that generalizes all examples by finding a description of the *target concept*. For supervised learning, the examples are labeled as positive or negative instances of the target concept by an oracle. The learnability of concepts written in (sub-)languages of the DL supported in the system Classic, which is an extension of $\mathcal{ALN}$, was investigated in [CH94a]. In particular, the PAC-learnability of these concepts was studied. PAC stands for probably approximately correct learning proposed by Vailant in [Vai84]. In this setting the training samples were either individuals [CH94b] or concepts [CH94a]. In the proposed learning algorithm the lcs was employed as a sub-procedure.

The method to generalize concepts from individuals was later re-discovered for the bottom-up approach to extend ontologies, which is our main application of the lcs.

**Similarity-based retrieval of information.** The lcs inference is used for similarity-based information retrieval. In [MHN98; MM98] methods to retrieve information based on user specified examples are proposed. Here, the query as well as the knowledge of the system is stored in the ABox. The user selects some individuals to query for similar items. The selected examples are turned into a concept and then generalized by means of the lcs, and the application ABox can be queried for instances of the obtained concept, which are the result of the retrieval query. Similar methods are employed in other application domains to facilitate retrieval. In [AH02] large data bases of text annotated images are provided with a hierarchy of image description to facilitate browsing in the data base by users.

In [HLRT02b; HLRT02a] the dynamic discovery of web services is considered. Here the task addressed is to find, for given client query and a set of available e-services described in an ontology, a subset of these services that cover the query best. To this end the authors propose the reasoning service of computing *best covers*. The intuitive notion of best covers is the following: given a concept description $E$ a best cover is a conjunction of concept descriptions $Q$ such that $E$ and $Q$ share commonalities while the concept descriptions for the parts that are not shared between $E$ and $Q$

are minimal w.r.t. the lexicographic order. These commonalities between $Q$ and $E$ are obtained by the lcs. By means of best cover the parts that are common to the query and the service description and the parts that are different can be identified. The latter can then be used to ask the user to refine or change the query. In fact, computing best covers is another instance of the rewriting framework for DLs discussed in Section 2.4.3.

**Support for other inferences.** In the CLASSIC system the lcs inference was employed as a 'weak form of disjunction'. The idea is to replace a disjunction by the lcs of its disjuncts a method called *knowledge base vivification* in [CBH92]. The main motivation for this was to avoid disjunction as source of complexity for reasoning and to arrive at a tractable DL. With modern DL reasoning systems at hand this is no longer the main motivation for the lcs. However, in [HM01a] it is argued that 'degenerated' concept hierarchies, i.e., containing concepts with very many sibling concepts, cause longer run-times for classification. Now, one could in principle employ the lcs to generate auxiliary concepts that do not introduce additional disjunction in the TBox to complement the concept hierarchy. These lcs concepts would only need to be generated once and be kept for following classifications.

Besides for standard inferences, the lcs inference is used in many other NSIs as a sub-task. In concept matching the lcs is employed to obtain the valid substitution for each variable in the concept pattern [Küs01; Bra06]. Furthermore, the lcs is used as a sub-procedure in methods for computing concept approximations (an inference that we investigate thoroughly in Chapter 4), in the computation of the semantic concept difference [Tee94] and in the above mentioned computation of best covers [HLRT02a]. The use of the lcs as a sub-procedure of other reasoning tasks makes it desirable to extend the computation methods of this inference to more expressive DLs and, moreover, to have efficient implementations of the lcs.

### 3.1.2 Methods for computing the lcs

The lcs inference is investigated for a small range of DLs. For those DLs where it is known that the lcs always exists, computation methods have been devised. All algorithms for computing the lcs are structural algorithms, i.e. they proceed by traversing the inductive structure of the input concept descriptions and comparing their parts. The normalization of each concept description is the crucial step to have all the implicit information captured in the concept description (or possibly the TBox) made explicit for the structural comparison. The computation methods for the lcs rely on structural characterizations of subsumption. A sound and complete characterization of subsumption ensures that the normal form contains all implied facts. These characterizations as well as the computation algorithms for the lcs fall into two categories according the underlying representation they use: the automata-based approach and the graph-based approach. The former is typically employed for DLs that do not provide existential restrictions, while the latter can handle also the combination of existential and value restrictions.

### Automata-theoretic approach

For the DL $\mathcal{ALN}$ w.r.t. cyclic TBoxes, an automata-based algorithm for computing the lcs was devised in [Küs98]. Initially the automata-based approach was introduced in [Baa96] to characterize the semantics of cyclic terminologies for the DL $\mathcal{FL}_0$. The approach was then extended to $\mathcal{ALN}$ w.r.t. cyclic TBoxes in [Küs98].

Intuitively, the idea of this approach is to use finite automata that represent the concept descriptions and the underlying TBox. The concept descriptions from the TBox are first normalized by applying the rule

$$\forall R.(C \sqcap D) \rightarrow \forall R.C \sqcap \forall R.D$$

exhaustively. This form allows to sort all (possibly nested) value restrictions w.r.t. the concept name or number restriction they end with into so-called *value-restriction sets*, which are finite regular sets of words over the alphabet of role names: $V_C(P) := \{W \in N_R^* \mid C \sqsubseteq \forall W.P\}$. A value-restriction set of $C$ for $P$ is a set of role paths to a concept name $P$ or a number restriction $P$ that is subsumed by $C$. Thus the value restriction set contains not only those role paths that appear in $C$ syntactically. Subsumption is characterized by inclusion between these languages and the lcs is obtained by intersection of value restriction sets of the input concept descriptions. The $\mathcal{ALN}$-lcs w.r.t. cyclic TBoxes can be computed in exponential time [Küs98].

The automata-based approach has been employed for unfoldable TBoxes by us in [BTK03b], where a lcs computation algorithm for the DL $\mathcal{FL}_0^+$ was devised. $\mathcal{FL}_0^+$ allows conjunction and value restriction for (possibly) transitive roles. For $\mathcal{FL}_0^+$, the representation of concept descriptions by formal languages could be extended by means of an operator for the transitive closure of formal languages. Computing the lcs of $\mathcal{FL}_0^+$-concept descriptions can be done in polynomial time.

### Graph-based approach

The graph-based approach is applied to a wider range of DLs. Here, concept descriptions are represented as description graphs, in which nodes are labeled with concept names and edges with role names and, in case existential *and* value restrictions are present in the DL, with the respective quantor. The characterization of subsumption is then given by homomorphisms between description graphs. Now, the method for computing the lcs for the different DLs works in the following basic steps:

1. Extract implicit information in each of the input concept descriptions and make it explicit by adding it to the concept description.

2. Represent each of the obtained concept descriptions in a description graph.

3. Compute the graph product of the description graphs of the input concepts and read the concept description from the obtained graph.

In case the lcs is computed for concept descriptions or unfoldable TBoxes, the concept descriptions can be represented by acyclic graphs.

The first lcs computation algorithm described in the literature was in fact a graph-based algorithm [CBH92]. This computation algorithm was tailored to the DL $\mathcal{ALNS}$ which the system CLASSIC offered, which extends $\mathcal{ALN}$ by attributes and attribute chain agreements. *Attribute chain agreements* allow to enforce that individuals reachable via two different attribute paths coincide. The algorithm proposed for this in [CBH92] turned out to be incomplete. In [Küs01] a complete algorithm for $\mathcal{ALNS}$ was devised. However, it was shown in [KB01] that depending on whether attributes are interpreted as total or partial functions the lcs may or may not exist. In case attributes are interpreted as partial function, the lcs for $\mathcal{ALNS}$ always exists and can be computed for two concept descriptions in polynomial time, while the lcs of $n > 2$ concept descriptions can grow exponentially in the size of the input concepts.

The graph-based lcs computation algorithm for $\mathcal{ALE}$-concept descriptions (and relevant sub-languages: $\mathcal{FLE}$ and $\mathcal{EL}$) is given in [BKM99]. Here, the concept descriptions are represented as concept *trees* and the characterization of subsumption is given by homomorphism between $\mathcal{ALE}$-description trees. The lcs can be directly obtained by the cross-product of $\mathcal{ALE}$-description trees. In the following sections of this chapter we discuss the results for $\mathcal{ALE}$ presented in [BKM99] extensively as well as the results for the computation of the lcs in $\mathcal{ALEN}$, i.e., in the presence of number restrictions in the language, see [KM01b]. The formal preliminaries introduced in these papers and the characterization of subsumption in particular, are the basis for our investigations on concept approximation and difference in Chapter 4. While the lcs in $\mathcal{ALE}$ can grow exponentially in size of the input concept descriptions, the lcs in $\mathcal{ALEN}$ can grow double exponential. Strictly speaking, the approach for $\mathcal{ALEN}$ from [KM01b] is not a graph-based approach, since it operates on $\mathcal{ALEN}$-concept descriptions directly. We list it here for the sake of completeness.[10]

A notable case of the graph-based approach is the computation algorithm for $\mathcal{FLE}^+$-concept descriptions. This DL augments $\mathcal{FLE}$ with transitive roles. Our results presented in [BTK03a; BT03] include lcs computation algorithms for $\mathcal{FLE}^+$, $\mathcal{EL}^+$ and $\mathcal{ELH}^+$ ($\mathcal{EL}$ with transitive roles and role hierarchies respectively). To accommodate the graph-based approach for transitive roles, directed acyclic graphs (DAGs) have to be used to handle role edges following from transitivity of roles. Furthermore, for the generation of DAGs for $\mathcal{FLE}^+$ concept descriptions, blocking conditions are required to ensure termination. Before generating a new node in the DAG, the blocking conditions test whether a node with the required label already exists among the predecessors of the current node, and if so, reuses this node instead of generating a new one.

Moreover, in case of $\mathcal{FLE}^+$, the product graph of $\mathcal{FLE}^+$-description graphs can become cyclic. In contrast to most other cases where the graph-based approach is used, this necessitates a non-trivial procedure for 'reading out' $\mathcal{FLE}^+$-concept descriptions from the $\mathcal{FLE}^+$-product graph. To sum up, the extension of the graph-based approach to transitive roles is rather involved from a conceptual point of view. A detailed discussion of this lcs computation algorithm is beyond the scope of this the-

[10] We are aware of the work presented in [Man01], where a similar algorithm for $\mathcal{ALEQ}$ ($\mathcal{ALE}$ extended by qualified number restrictions) is investigated. However, the proposed algorithm is already incorrect for the sub-language $\mathcal{ALEN}$ due to incorrect treatment of the interactions of at-most and value restrictions, as the discussion in [KM01b] reveals.

| | binary lcs | | $n$-ary lcs | |
|---|---|---|---|---|
| $\mathcal{EL}$ | P | [BKM99] | ExpTime | [BKM99] |
| $\mathcal{ALE}$ | ExpTime | [BKM99] | ExpTime | [BKM99] |
| $\mathcal{ALEN}$ | 2-ExpTime | [KM01b] | 2-ExpTime | [KM01b] |
| $\mathcal{ALNS}$ | P | [Küs01] | ExpTime | [Küs01] |

Table 3.1: Complexity of computing the lcs of concept descriptions.

sis. For $\mathcal{FLE}^+$, $\mathcal{EL}^+$ and $\mathcal{ELH}^+$ the devised algorithms in [BTK03a; BT03] can yield concept descriptions for the lcs that are exponential in the size of the input concept descriptions.

An overview of complexity results of the lcs computation algorithms for concept descriptions mentioned in this section are displayed in Table 3.1. The table lists the complexities depending on whether the lcs is applied to two concept descriptions or to a sequence of $n$ concept descriptions with $n > 2$.

The graph-based approach is also employed for computing the lcs of $\mathcal{EL}$-concept descriptions w.r.t. cyclic TBoxes in [Baa03b; Baa03a]. Cyclic TBoxes can be interpreted w.r.t. different kinds of semantics. For the descriptive semantics—the standard semantics for DL systems—it has been shown in the above cited paper that the lcs does not need to exist. However, for greatest fixed-point semantics, it has been shown that the lcs always exists. For this kind of semantics the binary lcs can be computed in polynomial time and an optimal computation algorithm based on product graphs has been presented. The algorithm uses simulations as a means to characterize subsumption between description graphs instead of homomorphisms. It has been shown in [Baa03c] that subsumption w.r.t. cyclic $\mathcal{EL}$-TBoxes no longer corresponds to the existence of homomorphisms, but to the existence of simulations between $\mathcal{EL}$-description graphs. Furthermore, the existence of a simulation between graphs can be tested in polynomial time [HHK95], while testing the existence of a homomorphism between graphs is a NP-complete problem [GJ79].

The lcs algorithm for cyclic $\mathcal{EL}$-TBoxes is used in [Bra06] as a basis for a computation algorithm for $\mathcal{EL}$ w.r.t. to hybrid TBoxes. Intuitively a hybrid TBox consists of two parts: a general TBox $\mathcal{F}$ ('foundation') and a possibly cyclic TBox $\mathcal{T}$ ('terminology') defined over the same set of atomic concepts and roles. The different parts of the hybrid TBox are interpreted w.r.t. different semantics: the foundation $\mathcal{F}$ is interpreted by descriptive semantics while the terminology $\mathcal{T}$ is interpreted by greatest fixed-point semantics. It has been shown in the above cited thesis that the lcs for hybrid TBoxes can be reduced to lcs w.r.t. greatest fixed-point semantics in polynomial time. In this way the lcs can be computed in polynomial time in the size of the hybrid TBox. The $n$-ary lcs w.r.t. hybrid $\mathcal{EL}$-TBoxes can be computed in exponential time in the size of the input and is of exponential size in the size of the input in the worst-case, which has also been shown in [Bra06].

In this thesis we focus on unfoldable TBoxes instead of cyclic or general TBoxes. For our application of supporting naive users in extending their ontologies the choice of

unfoldable TBoxes is more faithful, since they are easier to comprehend. Furthermore, in the light of the negative result regarding the computation of the lcs in $\mathcal{EL}$ w.r.t. general TBoxes with descriptive semantics, general TBoxes do not seem to be a good starting point for investigating the computation of common subsumers in more expressive DLs. We employ mainly the description-based variant for the structural characterization of subsumption and for devising computation methods for the NSIs considered.

In the remainder of the chapter, we recall and discuss the above mentioned results on structural characterization of subsumption and on the computation of the lcs in detail—first in the DL $\mathcal{ALE}$ and then in $\mathcal{ALEN}$. These results form the basis for our investigations on concept approximation and the computation of common subsumers w.r.t. a background terminology.

## 3.2 The lcs in $\mathcal{ALE}$

The method for computing the lcs for $\mathcal{ALE}$-concept descriptions was proposed and proven correct in [BKM99]. For this DL the lcs always exists and it can be computed effectively using the graph-based approach. As a typical instance of this approach the algorithm proceeds in the three basic steps:

1. extraction of implicit information and adding it explicitly to the concept descriptions,
2. representation of the concepts as description trees, and
3. computation of the cross-product of the description trees, and reading the concept description from the cross-product tree.

Since the lcs operator is associative, the $n$-ary lcs operation can be realized by the successive application of the binary one. Thus it suffices to describe the binary operation, which we do in the following.

### 3.2.1 Normalization $\mathcal{ALE}$-concept descriptions

The extraction of implicit information is composed of several steps. If the input concept descriptions contain concept names that are defined in the TBox, the information from the TBox must be made explicit. To this end the concept descriptions are *unfolded* w.r.t. the TBox. The unfolding process replaces the concept names of defined concepts by their definition from the TBox in a 'macro-expand' fashion.

**Definition 17 (Unfolded concept description).** Let $C$ be a concept description and $\mathcal{T}$ be an unfoldable TBox. $C'$ is the *unfolded concept description* of $C$ if $C'$ is obtained by exhaustively replacing every concept name that is a defined concept w.r.t. $\mathcal{T}$ by its definition from $\mathcal{T}$. ◇

An unfolded concept description contains only concept names of primitive concepts. Obviously, this procedure would not necessarily terminate for cyclic TBoxes. In his

seminal paper [Neb88] Nebel examined the complexity of the unfolding procedure. The unfolding of concept descriptions for DLs that contain existential as well as value restrictions can grow exponentially in the size of the initial concept description. Thus this step is a source of complexity for the lcs procedure, if the lcs is computed w.r.t. a TBox.

The step of unfolding the concept descriptions w.r.t. a TBox means to write the information from the terminology directly into the concept descriptions and to be able to ignore the concept definitions from TBox for the rest of the computation.[11]

### $\mathcal{ALE}$-Normalization

Next, each concept description is transformed into a normal form. The idea is to arrive at a representation of the concept description that makes all the consequences of interactions between the concept constructors explicit. In case the respective DL allows to express inconsistencies[12] the normal form also propagates inconsistencies. Next, we introduce the normal form for $\mathcal{ALE}$-concept descriptions and to that end we need the notion of role depth and role level.

**Definition 18 (Role depth, role level).** For a concept description $C$ the *role depth* $rd(C)$ is inductively defined as follows:

$$\begin{aligned} rd(N) &:= 0 && \text{, where } N \in N_C \cup \{\bot, \top\} \\ rd(\neg C) &:= rd(C) \\ rd(\delta\ n\ r) &:= 0 && \text{, where } \delta \in \{\leq, \geq\} \\ rd(C_1\ \rho\ C_2) &:= \max\{rd(C_1), rd(C_2)\} && \text{, where } \rho \in \{\sqcap, \sqcup\} \\ rd(Qr.C) &:= 1 + rd(C) && \text{, where } Q \in \{\exists, \forall\} \end{aligned}$$

A *role level* of a concept $C$ is the set of all concept descriptions occurring on the same role depth in $C$. The top most role level of a concept description is also called its *top-level*. ◇

We assume that $N_R = \{r\}$, in order to keep the following definitions simple. Now, the $\mathcal{ALE}$-normal form is computed by removing concept descriptions equivalent to $\top$, replacing inconsistent concept descriptions by $\bot$, joining value restrictions for the same role, and propagating value restrictions into existential restrictions on all role levels.

**Definition 19 ($\mathcal{ALE}$-normal form).** Let $E, F$ be $\mathcal{ALE}$-concept descriptions. An $\mathcal{ALE}$-concept description $C$ is in *$\mathcal{ALE}$-normal form* iff all of the following rules have been

[11] Besides for some subsumption tests in the course of the lcs computation as we will see.
[12] For instance, the DLs $\mathcal{EL}$ and $\mathcal{FLE}$ cannot express inconsistencies.

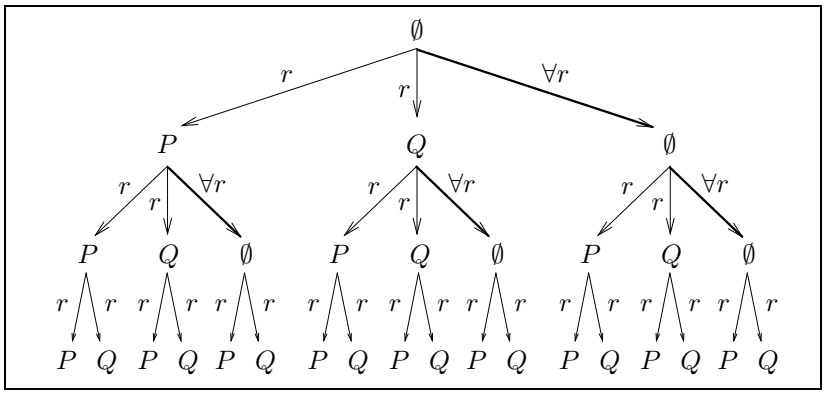


Figure 3.1: The $\mathcal{ALE}$-description tree of the normal form of $C_3$ from Example 20.

applied exhaustively (modulo associativity and commutativity of conjunction):

$$\forall r.\top \longrightarrow \top, \tag{3.1}$$
$$E \sqcap \top \longrightarrow E, \tag{3.2}$$
$$A \sqcap \neg A \longrightarrow \bot \quad \text{for each } A \in N_C, \tag{3.3}$$
$$\exists r.\bot \longrightarrow \bot, \tag{3.4}$$
$$E \sqcap \bot \longrightarrow \bot, \tag{3.5}$$
$$\forall r.E \sqcap \forall r.F \longrightarrow \forall r.(E \sqcap F), \tag{3.6}$$
$$\forall r.E \sqcap \exists r.F \longrightarrow \forall r.E \sqcap \exists r.(E \sqcap F). \tag{3.7}$$

$\diamond$

The first two rules and rule 3.6 reduce redundancy in the concept description. The rules 3.3 to 3.5 propagate inconsistencies. Rule 3.7 propagates value restrictions onto existential restrictions. The last rule is a source of an exponential blow-up of the concept description [BKM99; BT01a]. The following example is a well-known example taken from [BKM99] that demonstrates this effect.

**Example 20.** *We define the following sequence $C_1, C_2, C_3, \ldots$ of $\mathcal{ALE}$-concept descriptions:*

$$C_n := \begin{cases} \exists r.P \ \sqcap \ \exists r.Q, & n = 1 \\ \exists r.P \ \sqcap \ \exists r.Q \ \sqcap \ \forall r.C_{n-1}, & n > 1. \end{cases}$$

*Obviously, the size of $C_n$ is linear in $n$. However, applying the $\mathcal{ALE}$-normalization rule 3.7 to $C_n$ yields a description of size exponential in $n$. If one ignores the value restrictions (and everything occurring below a value restriction), then the description tree corresponding to the normal form of $C_n$ is a full binary tree of depth $n$, where the nodes reached by going to the left are labeled with $P$ and the ones reached by going to the right are labeled with $Q$. Figure 3.1 shows the $\mathcal{ALE}$-description tree of the normal form of $C_3$.*

We introduce some *accessors* for parts of concept descriptions, which will help to formulate the reasoning algorithms. Let $C$ be a concept description:

$prim(C)$ denotes the set of all (possibly negated) concept names and the bottom-concept occurring on the top role level of $C$;

$\mathsf{val}_r(C) := C_1 \sqcap \cdots \sqcap C_n$, if there exist value restrictions of the form $\forall r.C_1, \ldots, \forall r.C_n$ on the top role level of $C$; otherwise, $\mathsf{val}_r(C) := \top$;

$\mathsf{ex}_r(C) := \{C' \mid \text{there exists } \exists r.C' \text{ on the top role level of } C\}$;

Equipped with these accessors we can represent every $\mathcal{ALE}$-concept description as:

$$C = \prod_{D \in prim(C)} D \;\sqcap \prod_{r \in N_R} \Big( \prod_{D \in \mathsf{ex}_r(C)} \exists r.D \;\sqcap\; \forall r.\mathsf{val}_r(C) \Big).$$

### 3.2.2 Characterization of subsumption in $\mathcal{ALE}$

The lcs computation algorithm for $\mathcal{ALE}$ and its proof for correctness build on structural characterization of subsumption. Both, the characterization and the lcs algorithm, are formulated based on the notion of $\mathcal{ALE}$-description trees. $\mathcal{ALE}$-description trees are a graph representation for $\mathcal{ALE}$-concept descriptions that were introduced in [BKM99] and are defined as follows.

**Definition 21 ($\mathcal{ALE}$-description tree).** An *$\mathcal{ALE}$-description tree* is a tree of the form $\mathcal{G} = (V, E, v_0, \ell)$ with root $v_0$ where

- the edges in $E$ are labeled with role names $r$ from $N_R$ or with $\forall r$ for some $r \in N_R$, and
- the nodes $v \in V$ are labeled with sets $\ell(v) = \{P_1, \ldots, P_n\}$ where each $P_i$, $1 \leq i \leq n$, is of one of the following forms: $P_i \in N_C$, $P_i = \neg P$ for some $P \in N_C$, or $P_i = \bot$.

The empty label corresponds to the top-concept. ◇

It has been shown in the above cited paper how to convert $\mathcal{ALE}$-concept descriptions into $\mathcal{ALE}$-description trees and vice versa. Subsumption between two $\mathcal{ALE}$-concept descriptions can now be characterized by finding a homomorphism between the $\mathcal{ALE}$-description trees of normalized $\mathcal{ALE}$-concept descriptions.

**Definition 22 (Homomorphisms between $\mathcal{ALE}$-description trees).** A *homomorphism* from an $\mathcal{ALE}$-description tree $\mathcal{H} = (V_H, E_H, w_0, \ell_H)$ to an $\mathcal{ALE}$-description tree $\mathcal{G} = (V_G, E_G, v_0, \ell_G)$ is a mapping $\varphi : V_H \longrightarrow V_G$ such that

1. $\varphi(w_0) = v_0$,
2. for all $v \in V_H$ we have $\ell_H(v) \subseteq \ell_G(\varphi(v))$ or $\ell_G(\varphi(v)) = \{\bot\}$,
3. for all $vrw \in E_H$, either $\varphi(v)r\varphi(w) \in E_G$, or $\varphi(v) = \varphi(w)$ and $\ell_G(\varphi(v)) = \{\bot\}$, and

4. for all $v \forall r w \in E_H$, either $\varphi(v) \forall r \varphi(w) \in E_G$, or $\varphi(v) = \varphi(w)$ and $\ell_G(\varphi(v)) = \{\bot\}$.

$\diamond$

The homomorphism ensures that for each node $v$ of the $\mathcal{ALE}$-description tree $\mathcal{H}$ there is a node $w$ in $\mathcal{G}$ with a label set that is a super-set of the node label of $v_i$. Furthermore the homomorphism ensures that for each node $v$ in $\mathcal{H}$ there is a node $w$ in $\mathcal{G}$, such that, all successor nodes $v_i$ of $v$ have a corresponding node $w_j$ in $\mathcal{G}$ such that $(v, v_i)$ and $(w, w_j)$ are connected via an edge with the same label and $(v_i, w_j) \in \varphi$. The idea for the characterization of subsumption is to embed the description tree of the subsumer concept in the description tree of the subsumee concept. This idea was formalized for $\mathcal{ALE}$ in the following theorem.

**Theorem 23 ([BKM99]).** *Let $C, D$ be $\mathcal{ALE}$-concept descriptions in $\mathcal{ALE}$-normal form and $\mathcal{G}_C, \mathcal{G}_D$ the corresponding $\mathcal{ALE}$-description trees. Then $C \sqsubseteq D$ iff there exists a homomorphism from $\mathcal{G}_D$ to $\mathcal{G}_C$.*

The proof of correctness of this theorem was also given in [BKM99]. In [Küs01] in addition a description-based characterization of subsumption in $\mathcal{ALE}$ is given, which can easily be derived from Theorem 23.

**Theorem 24 ([Küs01]).** *Let $C, D$ be $\mathcal{ALE}$-concept descriptions in $\mathcal{ALE}$-normal form. Then $C \sqsubseteq D$ iff $C \equiv \bot$ or*

1. *$prim(D) \subseteq prim(C)$;*
2. *for every $D' \in \mathsf{ex}_r(D)$, there exists an existential restriction in $C' \in \mathsf{ex}_r(C)$ s.t. $C' \sqcap \mathsf{val}_r(C) \sqsubseteq D'$, and*
3. *for every $r \in N_R$, $\mathsf{val}_r(C) \sqsubseteq \mathsf{val}_r(D)$.*

These characterizations are not meant to be proposals for efficient subsumption algorithms to be implemented, they merely serve as tools for proofs. However, bearing in mind that the normalization can cause an exponential blow-up of the concept size, it is not surprising that subsumption in $\mathcal{ALE}$ is in fact NP-complete [DLN+92].

We will mainly refer to the description-based characterization of subsumption in this thesis. We will use $\mathcal{ALE}$-description trees only as a graphic representation for illustration.

### 3.2.3 LCS algorithm for $\mathcal{ALE}$

In [BKM99] it was shown that the computation of the lcs in $\mathcal{ALE}$ can be done by computing the cross-product of the description trees of the normalized $\mathcal{ALE}$-concept descriptions and reading the concept description from it. Since the resulting graph is again an $\mathcal{ALE}$-description tree, the lcs concept description can be obtained directly from this tree. In addition to the cross-product computation a constructive algorithm on concept descriptions has been formulated. This algorithm is displayed in Figure 3.2 for $N_R = \{r\}$ for sake of simplicity. First the algorithm tests for trivial cases and, in

Let $C, D$ be two normalized $\mathcal{ALE}$-concept descriptions.
If $C \sqsubseteq D$, then $\mathsf{lcs}_{\mathcal{ALE}}(C, D) := D$,
if $D \sqsubseteq C$, then $\mathsf{lcs}_{\mathcal{ALE}}(C, D) := C$,
otherwise

$$\begin{aligned}\mathsf{lcs}_{\mathcal{ALE}}(C, D) \quad = \quad & \bigsqcap_{A \in prim(C) \cap prim(D)} A \quad \sqcap \\ & \bigsqcap_{C' \in \mathsf{ex}_r(C), D' \in \mathsf{ex}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{ALE}}(C', D') \quad \sqcap \\ & \forall r.\mathsf{lcs}_{\mathcal{ALE}}(\mathsf{val}_r(C), \mathsf{val}_r(D)).\end{aligned}$$

Figure 3.2: The lcs computation algorithm for $\mathcal{ALE}$-concept descriptions.

case the lcs is not trivial, it constructs the lcs concept description recursively. In this construction, the empty conjunction stands for the top-concept $\top$. The recursive calls of $\mathsf{lcs}_{\mathcal{ALE}}()$ are well-founded since the role depth of the concept descriptions decreases with each call.

The concept size of a normalized $\mathcal{ALE}$-concept description cannot be bounded by a polynomial function as we saw in Example 20. This is already the case for normalized $\mathcal{FLE}$-concept descriptions. Another source of complexity is the number of recursive calls for existential restrictions (or, if you like, the computation of the $r$-edge successors in the graph-based setting). In $\mathcal{EL}$ the size of the lcs of 2 input concepts is polynomial, while the concept size of the lcs of $n > 2$ $\mathcal{EL}$-concepts can grow exponentially in the size of the input. In [BKM98] an example of a sequence of $n$ $\mathcal{EL}$-concept descriptions with size linear in $n$ is given, for which the lcs is a full binary tree with depth $n$ and which illustrates the exponential blow-up by product computation.[13] To sum-up, the complexity of computing the lcs in $\mathcal{EL}$ is polynomial for 2 input concepts and exponential for more than 2. For $\mathcal{FLE}$ and $\mathcal{ALE}$ the computation of the lcs can yield concept descriptions exponentially larger than the input concept descriptions (see Table 3.1 on page 42). Given this, a natural question is whether the lcs concept descriptions can be represented in a more compact form. We address this question in Section 3.4.

## 3.3 The lcs in $\mathcal{ALEN}$

The DL $\mathcal{ALEN}$ augments $\mathcal{ALE}$ with number restrictions. For $\mathcal{ALEN}$-concept descriptions the lcs always exists and can be computed effectively. This result has been shown in [KM01b]. We discuss this algorithm for computing the lcs of $\mathcal{ALEN}$-concept descriptions here in detail in preparation for our investigations of concept approximation in

[13] We discuss this example later in detail in this chapter.

the presence of number restrictions presented in Section 4.2. The structural characterization of subsumption and the lcs algorithm for $\mathcal{ALEN}$ is given in the description-based way in [KM01b].

The application of structural algorithms requires that implicit information in the concept descriptions is made explicit. This kind of information is mainly implied by interaction of the different concept constructors present in the DL. In case of $\mathcal{ALEN}$ these interactions are far more complex than in the case of $\mathcal{ALE}$, since number restrictions can interact with existential and value restrictions as we shall see.

However, the computation of all information implied in $\mathcal{ALEN}$-concept descriptions already requires the application of the lcs. So, in order to decouple the proofs of soundness and correctness of the characterization of subsumption and of the lcs, the characterization of subsumption is given based on a weak normal form, that does not yet make all facts explicit. Nevertheless, we examine next how to extract induced information from $\mathcal{ALEN}$-concept descriptions to gain better understanding for the interactions of the concept constructors in $\mathcal{ALEN}$. The notions introduced next are used in the lcs computation algorithm.

### 3.3.1 Implicit information in $\mathcal{ALEN}$-concept descriptions

As usual, if we want to compute the lcs of concept descriptions that contain names of concepts defined w.r.t. a TBox, these concept descriptions must be unfolded first. Then the induced information can be computed. In case of $\mathcal{ALEN}$ the number restriction can imply consequences for the existential and value restrictions and vice versa. For instance, an existential restriction on role $r$ obviously implies $(\geq 1\ r)$, while an at-most restriction to, say $n$ role-successors, can force that $n+m$ explicitly mentioned existential restrictions have to be grouped to $n$ existential restrictions. To gain better understanding of the possible interactions between the concept constructors in $\mathcal{ALEN}$, we consider the following example:

**Example 25.** *Consider the $\mathcal{ALEN}$-concept description $C_{ex} \equiv C_{ex1} \sqcup C_{ex2}$ over the set of concept names $N_C := \{A_1, A_2, P, Q\}$ with*

$$C_{ex1} := \exists r.(P \sqcap A_1) \ \sqcap\ \exists r.(P \sqcap A_2) \ \sqcap\ \exists r.(\neg P \sqcap A_1) \ \sqcap\ \exists r.Q \ \sqcap\ (\leq 2\ r)$$
$$C_{ex2} := \forall r.(A_1 \sqcap A_2) \ \sqcap\ (\geq 1\ r).$$

*We want to compute the induced concept descriptions of $C_{ex1}$ and $C_{ex2}$.*

***Induced number restrictions.*** *As explicit number restrictions, we find $(\leq 2\ r)$ in $C_{ex1}$ and $(\geq 1\ r)$ in $C_{ex2}$. Since $C_{ex1}$ has existential restrictions with $P$ and another one with $\neg P$, we know that at least two distinct $r$-successors must exist. Thus, $C_{ex1}$* induces $(\geq 2\ r)$.

***Induced existential restrictions.*** *$C_{ex1}$ has 4 existential restrictions while the number of $r$-successors is limited to 2 by the at-most restriction. Hence, the 4 existential restrictions must be merged into 2 such that consistency is preserved. This*

*can be done in two ways, yielding the possibilities*

$$\exists r.(P \sqcap A_1 \sqcap A_2 \sqcap Q) \sqcap \exists r.(\neg P \sqcap A_1) \qquad \textit{or}$$
$$\exists r.(P \sqcap A_1 \sqcap A_2) \sqcap \exists r.(\neg P \sqcap A_1 \sqcap Q).$$

*Although $C_{ex2}$ has no explicit existential restrictions, the at-least restriction $(\geq 1\ r)$ implies one $r$-successor for which the value restriction holds. So $\exists r.(A_1 \sqcap A_2)$ is an induced $\mathcal{ALEN}$-concept description of $C_{ex2}$.*

***Induced value restrictions.*** *The concept $C_{ex1}$ has no explicit value restrictions. Nevertheless, as seen above, $C_{ex1}$ has exactly 2 $r$-successors and every consistent merging has $A_1$ in every existential restriction. Hence, $\forall r.A_1$ is induced as a value restriction for $C_{ex1}$.*

*Overall, we obtain the following induced concept descriptions:*

$$\begin{aligned} C_{ex1} \equiv\ & \exists r.(P \sqcap A_1) \ \sqcap\ \exists r.(P \sqcap A_2) \ \sqcap\ \exists r.(\neg P \sqcap A_1) \ \sqcap\ \exists r.Q \\ & \sqcap\ \forall r.A_1 \ \sqcap\ (\leq 2\ r) \ \sqcap\ (\geq 2\ r) \\ C_{ex2} \equiv\ & \exists r.(A_1 \sqcap A_2) \ \sqcap\ \forall r.(A_1 \sqcap A_2) \ \sqcap\ (\geq 1\ r). \end{aligned}$$

This example already shows that the computation of the induced information is conceptually much more complex in $\mathcal{ALEN}$ than in $\mathcal{ALE}$. Next we introduce the methods defined in [KM01b] to compute the induced information of $\mathcal{ALEN}$-concept descriptions.

### Induced number restrictions in $\mathcal{ALEN}$

Number restrictions can be induced by existential restrictions and value restrictions. The induced number restrictions are not determined syntactically, but by using subsumption. The induced at-least restriction is defined as: $(\geq\ \mathsf{min}_r(C)\ r)$, where

$$\mathsf{min}_r(C) := max\{k \mid C \sqsubseteq (\geq k\, r)\}.$$

Note that $\mathsf{min}_r(C)$ is always finite. Similarly, the induced at-most restriction is defined as: $(\leq\ \mathsf{max}_r(C)\ r)$, with

$$\mathsf{max}_r(C) := min\{k \mid C \sqsubseteq (\leq k\, r)\};$$

if there exists no $k$ with $C \sqsubseteq (\leq k\, r)$, then $\mathsf{max}_r(C) := \infty$.

The computation of induced number restrictions requires subsumption tests, which is a PSPACE-complete problem for $\mathcal{ALEN}$ [Hem01]. The number of these tests can be bounded polynomially in the size of $C$.

### Induced existential restrictions in $\mathcal{ALEN}$

We need to formalize the notion of merging existential restrictions, which we already encountered for $C_{\text{ex1}}$. This is done by so-called *existential mappings* $\alpha$. Intuitively, each $\alpha$ is one combination to group all explicit existential restrictions to exactly as

many $r$-successors as allowed by the induced at-most restriction for $r$ in $\mathsf{max}_r(C)$, in a consistent way. Formally, $\alpha$ is defined as

$$\alpha\colon \{1,\ldots,n\} \longrightarrow 2^{\{1,\ldots,m\}},$$

where $n := min\{\mathsf{max}_r(C), |\mathsf{ex}_r(C)|\}$ and $m := |\mathsf{ex}_r(C)|$. Moreover, for every $\alpha$ we want to enforce that no trivial $r$-successors ($\exists r.\top$) are produced and that every mapping $\alpha$ partitions the set $\mathsf{ex}_r(C)$ into $n$ non-empty sets. Furthermore, merging existential restrictions must not lead to inconsistencies. This leads to the following conditions on existential mappings $\alpha$:

1. $\alpha(i) \neq \emptyset$ for all $1 \leq i \leq n$;
2. $\bigcup_{1\leq i\leq n} \alpha(i) = \{1,\ldots,m\}$ and $\alpha(i) \cap \alpha(j) = \emptyset$ for all $1 \leq i < j \leq n$;
3. $\bigsqcap_{j\in\alpha(i)} C'_j \sqcap \mathsf{val}_r(C) \not\equiv \bot$ for all $1 \leq i \leq n$ with $\mathsf{ex}_r(C) = \{C'_1,\ldots,C'_m\}$.

As we saw in Example 25, there may be several of these mappings for one $\mathcal{ALEN}$-concept description. The set of all existential mappings on a concept description $C$ satisfying the above Conditions (1) to (3)—modulo permutations—is denoted by $\Gamma_r(C)$, where $\Gamma_r(C) := \emptyset$, if $\mathsf{ex}_r(C) = \emptyset$.

Given an existential mapping $\alpha$, the corresponding *set of merged concept descriptions* is denoted by

$$\mathsf{ex}_r(C)^\alpha := \{ \bigsqcap_{j\in\alpha(i)} C'_j \mid 1 \leq i \leq n\}.$$

Now, let $\Gamma_r(C)$ denote the set of all existential mappings satisfying the above conditions. For a given set of $k$ mapping functions $\Gamma_r(C) = \{\alpha_1,\ldots,\alpha_k\}$ the *set of induced existential restrictions* $\mathsf{ind\text{-}ex}_r(C)$ of $C$ is defined as follows:

- if $\mathsf{ex}_r(C) \neq \emptyset$, then
  $\mathsf{ind\text{-}ex}_r(C) := \big\{ \mathsf{lcs}_{\mathcal{ALEN}}\big(\{C_l \sqcap \mathsf{val}_r(C) \mid 1 \leq l \leq k\}\big) \mid C_j \in \mathsf{ex}_r(C)^{\alpha_j}, 1 \leq j \leq k\big\}$;
- if $\mathsf{ex}_r(C) = \emptyset$ and $\mathsf{min}_r(C) \geq 1$, then $\mathsf{ind\text{-}ex}_r(C) := \{\mathsf{val}_r(C)\}$;
- otherwise, $\mathsf{ind\text{-}ex}_r(C) := \emptyset$.

The idea behind the use of the lcs for the computation of $\mathsf{ind\text{-}ex}_r$ (C) is the following: If the existential restrictions have to be merged and *every* valid existential mapping $\alpha_i$ yields a concept description, say $C_i$ ($C_i \in \mathsf{ex}_r(C)^{\alpha_i}$), which implies $D$, then $\exists r.D$ is induced by $C$ in all existential mappings, i.e., it is independent of the grouping of the concept descriptions in the set of merged concept descriptions. In order to find the concepts commonly implied by all sets of merged concept descriptions, the lcs is applied to all information implied by the existential restrictions, the commonalities of the concepts obtained by the different existential mappings must be considered.

However, since the lcs is only applied to (conjunctions of) sub-concept descriptions of $C$, the number of lcs applications is bounded by the number of existential restrictions on each role level and by the role depth of $C$. Nevertheless, the use of the lcs for the

computation of induced information makes the proof of existence of the lcs in [KM01b] much more involved.

In Example 25, for $C_{\text{ex1}}$ the first case of the definition of $\mathsf{ind\text{-}ex}_r()$ applies. We have two existential mappings, say $\alpha_1$ and $\alpha_2$, with $\mathsf{ex}_r(C_{\text{ex1}})^{\alpha_1} = \{P \sqcap A_1 \sqcap A_2 \sqcap Q, \neg P \sqcap A_1\}$ and $\mathsf{ex}_r(C_{\text{ex1}})^{\alpha_2} = \{P \sqcap A_1 \sqcap A_2, \neg P \sqcap A_1 \sqcap Q\}$. Now, the induced existential mapping is what is implied by all valid mappings. Extracting the commonalities of all valid existential mappings yields:

$$\begin{aligned}\mathsf{ind\text{-}ex}_r(C_{\text{ex1}}) = \big\{&\mathsf{lcs}_{\mathcal{ALEN}}\big(\{P \sqcap A_1 \sqcap A_2 \sqcap Q,\ P \sqcap A_1 \sqcap A_2\}\big),\\ &\mathsf{lcs}_{\mathcal{ALEN}}\big(\{P \sqcap A_1 \sqcap A_2 \sqcap Q,\ \neg P \sqcap A_1 \sqcap Q\}\big),\\ &\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\neg P \sqcap A_1,\ P \sqcap A_1 \sqcap A_2\}\big),\\ &\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\neg P \sqcap A_1,\ \neg P \sqcap A_1 \sqcap Q\}\big)\big)\\ = \{&P \sqcap A_1 \sqcap A_2,\ A_1 \sqcap Q,\ A_1,\ \neg P \sqcap A_1\}.\end{aligned}$$

For $C_{\text{ex2}}$, the second case of the definition of $\mathsf{ind\text{-}ex}_r()$ applies: $\mathsf{ind\text{-}ex}_r(C_{\text{ex2}}) = \{(A_1 \sqcap A_2)\}$.

### Induced value restrictions in $\mathcal{ALEN}$

New value restrictions can only be induced for two reasons. First, if $\mathsf{max}_r(C) = 0$, then $C \sqsubseteq \forall r.\bot$ holds. Second, the merging of existential restrictions in combination with at-most restrictions may induce value restrictions—as for $C_{\text{ex1}}$ in Example 25. This is the case, if the number in the induced at-most restriction $\mathsf{max}_r$ (C) and the minimal number $n$ for which an existential mapping can be found coincide. Intuitively, in this case *all* role-successors are 'known'. The minimal number of $r$-successors induced by incompatible existential restrictions, which is the number of role-successors in the strongest merging obeying the conditions for a valid existential mapping, is defined as:

$$\kappa_r(C) := \begin{cases} \mathsf{min}_r(\forall r.\mathsf{val}_r(C) \ \sqcap\ \bigsqcap_{C' \in \mathsf{ex}_r(C)} \exists r.C') & \text{if } \mathsf{ex}_r(C) \neq \emptyset,\\ 0 & \text{otherwise.}\end{cases}$$

To obtain $\kappa_r(C)$, the value restrictions from $C$ need to be propagated onto the existential restrictions. Now, only if $\kappa_r(C) = \mathsf{max}_r(C)$, value restrictions can be induced, since only then we 'know' all $r$-successors of instances of $C$. In Example 25, we encounter this case for $C_{\text{ex2}}$, since $\kappa_r(C_{\text{ex2}}) = \mathsf{max}_r(C_{\text{ex2}}) = 2$.

With all merged existential restrictions obtained from all existential mappings $\alpha \in \Gamma_r(C)$ collected in the set

$$\mathsf{ex}_r(C)^* := \bigcup_{\alpha \in \Gamma_r(C)} \mathsf{ex}_r(C)^\alpha$$

the induced value restriction $\mathsf{ind\text{-}val}_r(C)$ of $C$ is defined as follows:

- if $\mathsf{max}_r(C) = 0$, then $\mathsf{ind\text{-}val}_r(C) := \bot$;
- if $0 < \kappa_r(C) < \mathsf{max}_r(C)$, then $\mathsf{ind\text{-}val}_r(C) := \mathsf{val}_r(C)$;

- if $0 < \kappa_r(C) = \mathsf{max}_r(C)$, then
  $\mathsf{ind\text{-}val}_r(C) := \mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{val}_r(C) \sqcap C' \mid C' \in \mathsf{ex}_r(C)^*\})$

Again we have to employ the lcs to compute induced information to find out the commonalities of each existential restriction (with the explicitly mentioned value restriction propagated onto it) obtained from all valid existential mappings. In this application the number of lcs calls during the computation of $\mathsf{ind\text{-}val}_r(C)$ is bounded by the role depth of $C$.

For $\mathcal{ALEN}$ only a weak normal form is defined in [KM00], which replaces number restrictions by the induced number restrictions and conjoins value restrictions in each conjunction of an $\mathcal{ALEN}$-concept description. In fact, no 'strong normal form', where all induced information is explicitly expressed as an $\mathcal{ALEN}$-concept description exists, since the alternative ways of merging existential restrictions would have to be captured.

### 3.3.2 Structural characterization of subsumption in $\mathcal{ALEN}$

In [KM01b] an algorithm for structural subsumption of $\mathcal{ALEN}$-concept descriptions was given and proven to be sound and complete in [KM00]. Subsumption between $\mathcal{ALEN}$-concept descriptions can be tested by the conditions stated in the following theorem.[14]

**Theorem 26 ([KM01b]).** *Let $C, D$ be two $\mathcal{ALEN}$-concept descriptions with $\mathsf{ex}_r(C) = \{C_1, \ldots, C_m\}$. Then $C \sqsubseteq D$ iff $C \equiv \top$, $D \equiv \bot$ or all of the following holds:*

1. *$prim(D) \subseteq prim(C)$;*
2. *$\mathsf{max}_r(C) \leq \mathsf{max}_r(D)$;*
3. *$\mathsf{min}_r(C) \geq \mathsf{min}_r(D)$;*
4. *for all $D' \in \mathsf{ex}_r(D)$ it holds that*
   (a) *$\mathsf{ex}_r(C) = \emptyset, \mathsf{min}_r(C) \geq 1$, and $\mathsf{val}_r(C) \sqsubseteq D'$; or*
   (b) *$\mathsf{ex}_r(C) \neq \emptyset$ and for each $\alpha \in \Gamma_r(C)$, there exists $C' \in \mathsf{ex}_r(C)^\alpha$ such that $C' \sqcap \mathsf{val}_r(C) \sqsubseteq D'$ and*
5. *if $\mathsf{val}_r(D) \not\equiv \top$, then*
   (a) *$\mathsf{max}_r(C) = 0$; or*
   (b) *$\kappa_r(C) < \mathsf{max}_r(C)$ and $\mathsf{val}_r(C) \sqsubseteq \mathsf{val}_r(D)$; or*
   (c) *$0 < \kappa_r(C) = \mathsf{max}_r(C)$ and for all $C' \in \mathsf{ex}_r(C)^* : \mathsf{val}_r(C) \sqcap C' \sqsubseteq \mathsf{val}_r(D)$.*

The first three conditions of the structural subsumption algorithm are straightforward, but the conditions for existential and value restrictions are less obvious. By Condition 4 it is guaranteed that each existential restriction of the subsumer $D$ is more general than either the existential restriction implied by an at-least restriction

[14] Recall that we assumed $N_R = \{r\}$.

Let $C, D$ be two $\mathcal{ALEN}$-concept descriptions.
If $C \sqsubseteq D$, then $\mathsf{lcs}_{\mathcal{ALEN}}(C, D) := D$,
if $D \sqsubseteq C$, then $\mathsf{lcs}_{\mathcal{ALEN}}(C, D) := C$,
otherwise

$$\begin{aligned}\mathsf{lcs}_{\mathcal{ALEN}}(C, D) \;:=\; & \prod_{Q \in prim(C) \cap prim(D)} Q \;\sqcap \\ & (\leq max\{\mathsf{max}_r(C), \mathsf{max}_r(D)\}\; r) \;\sqcap \\ & (\geq min\{\mathsf{min}_r(C), \mathsf{min}_r(D)\}\; r) \;\sqcap \\ & \prod_{C' \in \mathsf{ind\text{-}ex}_r(C), D' \in \mathsf{ind\text{-}ex}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{ALEN}}(C', D') \;\sqcap \\ & \forall r.\mathsf{lcs}_{\mathcal{ALEN}}(\mathsf{ind\text{-}val}_r(C), \mathsf{ind\text{-}val}_r(D)),\end{aligned}$$

Where

- $(\leq\ max\{\mathsf{max}_r(C), \mathsf{max}_r(D)\}\; r)$ is omitted, if $\mathsf{max}_r(C) = \infty$ or $\mathsf{max}_r(D) = \infty$, and
- $\prod_{C' \in \mathsf{ind\text{-}ex}_r(C), D' \in \mathsf{ind\text{-}ex}_r(D)} \exists r.\mathsf{lcs}_{\mathcal{ALEN}}(C', D') := \top$, if $\mathsf{ind\text{-}ex}_r(C) = \emptyset$ or $\mathsf{ind\text{-}ex}_r(D) = \emptyset$.

Figure 3.3: The lcs computation algorithm for $\mathcal{ALEN}$-concept descriptions.

in combination with a value restriction in the subsumee $C$ (Condition 4a) or than one existential restriction from each existential mapping of the $C$ (Condition 4b). Condition 5 treats value restrictions for which three cases have to be distinguished: either the subsumee $C$ has no $r$-successor and thus $\forall r.\bot$ as a value restriction (Condition 5a) or there are no induced value restrictions in $C$ and thus there has to be a subsumption relation between the explicit value restrictions (Condition 5b) or, in case that the minimal number of $r$-successors for instances of $C$ ($\kappa_r(C)$) coincides with the number from the at-most restriction in $C$ there is an implicit value restriction for $C$.

The rather involved proof for completeness and correctness of the structural characterization of subsumption for $\mathcal{ALEN}$-concept descriptions is given in [KM01b]. We require this characterization as a formal basis for algorithms to be introduced in the following chapters.

### 3.3.3 LCS algorithm for $\mathcal{ALEN}$

The algorithm for computing the lcs of two $\mathcal{ALEN}$-concept descriptions as defined in [KM01b] is shown in Figure 3.3. We use a slightly more handy notation here than in the original paper based on the notions introduced in Section 3.3.1. Similar to the

algorithm for computing the lcs for $\mathcal{ALE}$ concept descriptions, the algorithm for $\mathcal{ALEN}$ constructs the lcs concept description recursively by treating the concept constructors in $\mathcal{ALEN}$ separately.

For the algorithm depicted in Figure 3.3 it has been shown in [KM00] that the lcs of a set of $\mathcal{ALEN}$-concept descriptions can be computed in double-exponential time in the size of the input concept descriptions. A corresponding lower bound is not known. However, it is clear from the results for the lcs in $\mathcal{ALE}$ that the lcs in $\mathcal{ALEN}$ needs at least exponential time in the worst case.

## 3.4 On the compact representation of least common subsumers

The concept description returned by the lcs for $\mathcal{ALE}$ can grow exponentially in the size of the unfolded input concepts in the worst case. Although these cases are not necessarily encountered in practice, it is clear that the lcs concept descriptions can grow too large to be comprehensible for a human reader.

In [BK00] *minimal rewritings* have been proposed to remedy this. Minimal rewritings replace sub-concept descriptions of a concept with equivalent concept names from the TBox (see Definition 15 on page 34). With computing the minimal rewriting of the lcs concept description as a 'post-processing' step, one can obtain smaller, but equivalent lcs concept descriptions.

Another approach to arrive at a more compact representation of lcs concept descriptions is to extend the TBox by new definitions and use these auxiliary concepts for structure sharing in the lcs concept, which was explored by us in [BT01a; BT02a]. More formally, the addressed problem can be stated as: Let $\mathcal{L}$ be a DL for which the lcs operation is exponential. Given input descriptions $C_1, \ldots, C_n$ with lcs $D$, does there always exist a TBox $\mathcal{T}$ whose size is polynomial in the size of $C_1, \ldots, C_n$ and a defined concept name $A$ in $\mathcal{T}$ such that $A \equiv_{\mathcal{T}} D$, i.e., the TBox defines $A$ such that it is equivalent to the lcs $D$ of $C_1, \ldots, C_n$?

We first examine the worst case examples for the lcs in $\mathcal{EL}$ and in $\mathcal{ALE}$. Encouragingly, these examples can be represented by introducing auxiliary concepts in the TBox. However, this technique cannot be used to compress lcs concept descriptions in general as we will see later.

### 3.4.1 Worst case examples

The following example was given in [BKM99] and demonstrates that the lcs of $n$ $\mathcal{EL}$-concept descriptions of size linear in $n$ may be exponential in $n$.

**Example 27.** *We define for each $n \geq 1$ a sequence $\{C_1^n, \ldots, C_n^n\}$ of $\mathcal{EL}$-concept descriptions. For $n \geq 0$ let*

$$D_n := \begin{cases} \top, & n = 0 \\ \exists r.(P \sqcap Q \sqcap D_{n-1}), & n > 0 \end{cases}$$

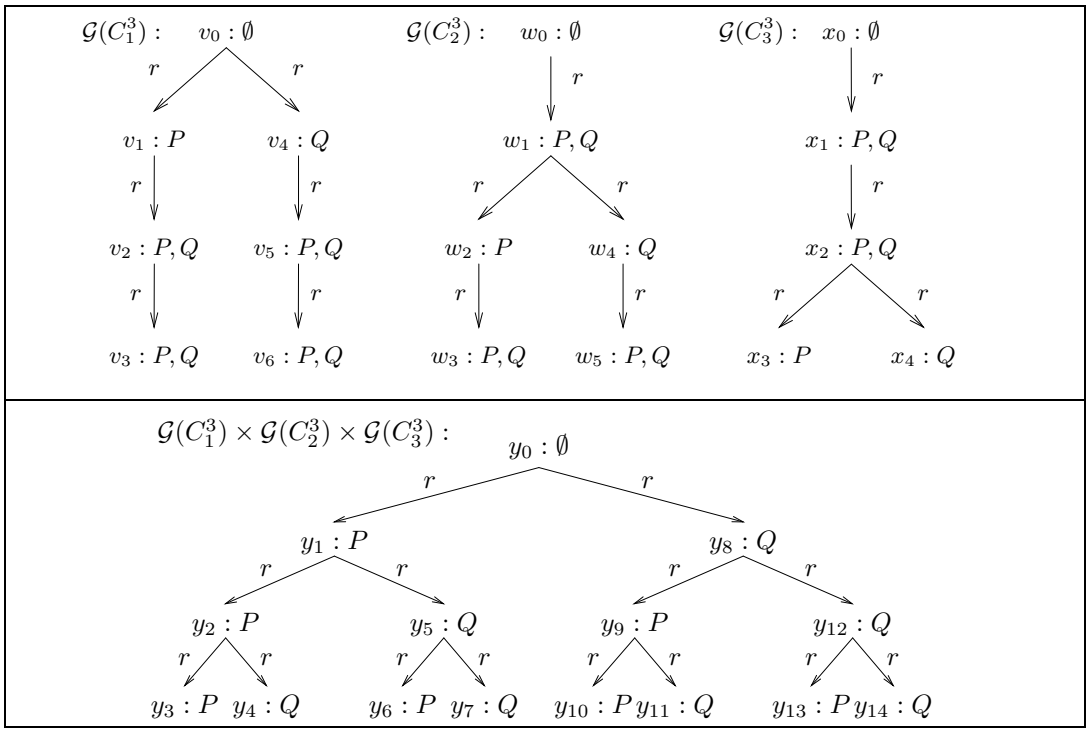


Figure 3.4: Description trees of $C_1^3, C_2^3, C_3^3$ and their product.

*and for* $n \geq 1$ *and* $1 \leq i \leq n$ *we define*

$$C_i^n := \begin{cases} \exists r.(P \sqcap D_{n-1}) \sqcap \exists r.(Q \sqcap D_{n-1}), & i = 1 \\ \exists r.(P \sqcap Q \sqcap C_{i-1}^{n-1}), & 1 < i \leq n. \end{cases}$$

*It is easy to see that each* $C_i^n$ *is linear in the size of* $n$*. The product of the corresponding description trees is a full binary tree of depth* $n$*, where the nodes reached by going to the left are labeled with* $P$ *and the ones reached by going to the right are labeled with* $Q$*. Obviously, the size of this tree is exponential in* $n$*. What is less obvious, but can also be shown (see [BKM99]), is that there is no smaller description tree representing the same concept (modulo equivalence).*

The lower half of Figure 3.4 depicts the tree obtained as the product of the description trees corresponding to the descriptions $C_1^3, C_2^3, C_3^3$. This tree is a full binary tree of depth 3, where the nodes reached by going to the left are labeled with $P$ and the ones reached by going to the right are labeled with $Q$.

Let us consider again Example 20 from page 45. As we can see in Figure 3.1, the exponentially large normal form constructed in this example has as its description tree the full binary tree of depth $n$, where the nodes reached by going to the left were labeled with $P$ and the ones reached by going to the right were labeled with $Q$. This concept can be defined in a TBox of size linear in $n$.

**Example 28.** *For $n \geq 1$, we consider the concept descriptions $C_n$ introduced in Example 20 and the concept descriptions $D_n$ defined in Example 27. By building the product of the description trees corresponding to the normal forms of $C_n$ and $D_n$, one basically removes the value restrictions from the normal form of $C_n$. Thus, one ends up with a lcs $E_n$ that agrees with the binary tree we obtained in Example 27. Again, it can be shown that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs.*

### 3.4.2 Using TBoxes to compress the lcs

The exponentially large lcs $E_n$ constructed in Examples 27 and 28 has as its description tree the full binary tree of depth $n$, where the nodes reached by going to the left were labeled with $P$ and the ones reached by going to the right were labeled with $Q$. This concept can be defined in a TBox of size linear in $n$.

**Example 29.** *Consider the following TBox $\mathcal{T}_n$:*

$$\{A_1 \doteq \exists r.P \sqcap \exists r.Q\} \ \cup$$
$$\{A_i \doteq \exists r.(P \sqcap A_{i-1}) \sqcap \exists r.(Q \sqcap A_{i-1}) \mid 1 < i \leq n\}.$$

*It is easy to see that the size of $\mathcal{T}_n$ is linear in $n$ and that $A_n \equiv_{\mathcal{T}_n} E_n$, i.e., the TBox $\mathcal{T}_n$ provides us with a compact representation of $E_n$.*

In general, however, such a compact representation by structure sharing is not possible. We will first give a counterexample for the $n$-ary lcs in $\mathcal{EL}$, and then for the binary lcs in $\mathcal{ALE}$. The main idea underlying both counterexamples is to generate description trees having exponentially many leaves that are all labeled by sets of concept names that are incomparable w.r.t. set inclusion. To this purpose, we consider the set of concept names $N_n := \{A_j^0, A_j^1 \mid 1 \leq j \leq n\}$, and define $A^{\mathbf{i}} := A_1^{i_1} \sqcap \ldots \sqcap A_n^{i_n}$ for each $n$-tuple $\mathbf{i} = (i_1, \ldots, i_n) \in \{0,1\}^n$.

**Counterexample for $\mathcal{EL}$.** For all $n \geq 1$ we define a sequence $C_1, \ldots, C_n$ of $n$ $\mathcal{EL}$-concept descriptions whose size is linear in $n$:

$$C_j := \exists r. \prod_{B \in N_n \setminus \{A_j^0\}} B \quad \sqcap \quad \exists r. \prod_{B \in N_n \setminus \{A_j^1\}} B.$$

Since each of the concepts $C_j$ contains two existential restrictions, the lcs of $C_1, \ldots, C_n$ contains $2^n$ existential restrictions. The concept descriptions occurring under these restrictions are obtained by intersecting the corresponding concept descriptions under the existential restrictions of the concept descriptions $C_j$. It is easy to see that these are exactly the $2^n$ concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$ introduced above. Since the descriptions $A^{\mathbf{i}}$ are pairwise incomparable w.r.t. subsumption, it is clear that there is no smaller $\mathcal{EL}$-concept description equivalent to this lcs. We show now that a TBox cannot be used to obtain a smaller representation of this concept description.

Obviously, to get a more compact representation of the lcs using a TBox, one needs duplication of concept names on the right-hand sides of the TBox. During unfolding of the TBox, this would, however, lead to duplication of sub-concepts. Since the

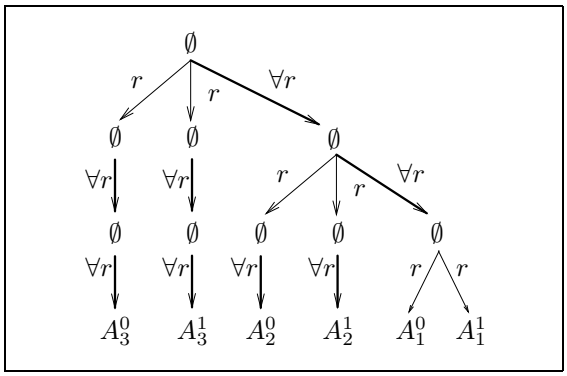


Figure 3.5: The $\mathcal{ALE}$-description tree corresponding to $C_3$.

(description tree of the) lcs we have constructed here has $2^n$ different leaves, such duplication does not help, since it can only duplicate leaves with the same label, but not generate leaves with different labels. Thus, in general, we cannot represent the lcs in a more compact way by introducing new definitions in an $\mathcal{EL}$ TBox.

**Counterexample for $\mathcal{ALE}$.** For $n \geq 1$ we define concept descriptions $C_n$ of size quadratic in $n$. For $n \geq 1$, let $F_j^i := \forall r. \cdots \forall r. A_{j+1}^i$ be the concept description consisting of $j$ nested value restrictions followed by the concept name $A_{j+1}^i$. We define

$$C_1 := \exists r. A_1^0 \sqcap \exists r. A_1^1,$$
$$C_n := \exists r. F_{n-1}^0 \ \sqcap \ \exists r. F_{n-1}^1 \ \sqcap \ \forall r. C_{n-1} \qquad \text{for } n > 1.$$

Figure 3.5 shows the description tree corresponding to $C_3$.

Applying the normalization rule $\forall r. E \sqcap \exists r. F \longrightarrow \forall r. E \sqcap \exists r. (E \sqcap F)$ to $C_n$ yields a normalized concept description whose size is exponential in $n$. If one ignores the value restrictions (and everything occurring below them), then the description tree corresponding to this normal form of $C_n$ is a full binary tree of depth $n$ whose $2^n$ different leaves are labeled by the $2^n$ concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

Let $D_n := \exists r. \cdots \exists r. \bigsqcap_{B \in N_n} B$ be the concept description consisting of $n$ nested existential restrictions followed by the conjunction of all concept names in $N_n$. Again, by building the product of the description trees corresponding to the normal forms of $C_n$ and $D_n$, one basically removes the value restrictions from the normal form of $C_n$. Thus, the lcs corresponds to the full binary tree of depth $n$ whose leaves are labeled by the concept descriptions $A^{\mathbf{i}}$ for $\mathbf{i} \in \{0,1\}^n$.

By an argument similar to the one for $\mathcal{EL}$ one can show that there is no smaller $\mathcal{ALE}$-concept description equivalent to this lcs, and that a TBox cannot be used to obtain a smaller representation.

This finding for the $\mathcal{ALE}$-example carries directly over to the case of representing the lcs in $\mathcal{ALEN}$ more compactly by extending the TBox. The examples given above show that the exponential size of the lcs concept description cannot be avoided by employing structure sharing, i.e., replacing common substructures by a name of a defined concept.

## 3.5 Extending the lcs to more expressive DLs

In this chapter we have supplied an overview of applications of the lcs and of methods for the computation of the lcs in different DLs and TBox formalisms. We have examined the methods to compute the lcs in the DLs $\mathcal{ALE}$ and $\mathcal{ALEN}$ in detail.

The computation of the lcs is central to implement the ontology extension approaches described in Section 1.2.2. In particular, the bottom-up extension and the customization of background ontologies are built on the inference service of computing common subsumers.

For many applications more expressive DLs are used to model the application domain. In particular, most of these DLs are propositionally closed and thus can also express disjunction. For these DLs the simple application of the lcs does not yield a useful result for the extension of ontologies by the approaches described in the introduction. Computing the lcs of the concepts $C_1, \ldots, C_n$ in a DL offering disjunction results in the disjunction $C_1 \sqcup \cdots \sqcup C_n$ as their lcs. This result is correct, but not a good starting point for a naive user to understand and edit the concept description since it does not make the commonalities of the input concepts explicit.

Now, in order to assess whether the lcs concept description proposed to the modeler captures a certain characteristic of the intended concept, she needs to comprehend which information is captured in the concept description and which is not. Therefore the displayed concept description should be succinct to be easily comprehensible.

In addition to this, the modeler has to locate the part in the concept description (or in the ontology) where the relevant information is stated in order to edit the concept description competently, which is obviously not fulfilled by just enumerating the input concepts as done in the plain disjunction of a naive lcs.

Since many DLs in use for practical applications allow for disjunction—most notably the DLs underlying the standards of OWL Lite and OWL DL—new methods to handle disjunction in a 'meaningful way' in the computation of common subsumers are required. These methods must be based on well-defined reasoning services with sound and complete computation methods to ensure reliable and predictable behavior when implemented and put to practice. In this thesis we propose and investigate two approaches for this and supply methods for the computation of the underlying reasoning services for it.

### The approximation-based approach

In the approximation-based approach disjunction is eliminated, before applying the lcs. More precisely, the input concept descriptions for the lcs are first translated into a DL that does not offer disjunction and then their commonalities are extracted.

To this end we introduce *concept approximation*, which in general computes for a concept description in a DL $\mathcal{L}_1$ the closest concept description w.r.t. subsumption in a DL $\mathcal{L}_2$.[15] Now, since not all concept constructors of $\mathcal{L}_1$ are present in $\mathcal{L}_2$, some information of the initial concept cannot be captured in the concept approximation.

[15] Typically, $\mathcal{L}_1$ would be more expressive than $\mathcal{L}_2$.

But since the concept description obtained from concept approximation is the closest w.r.t. subsumption, the loss of information is minimized.

For our application, where we use concept approximation as a preprocessing step for computing the lcs, the interesting cases are those where $\mathcal{L}_1$ offers disjunction and $\mathcal{L}_2$ does not. In the preprocessing step each of the selected concept descriptions are approximated in a DL without disjunction and then the lcs of these concept descriptions is computed. In the next chapter we investigate two such pairs of DLs for concept approximation.

**Common subsumers w.r.t. a background terminology**

In this approach two kinds of terminologies are used in combination: the background terminology written in an expressive DL and the user terminology written in a less expressive DL. While the background DL provides disjunction, the user DL does not. The concept descriptions written in the user DL, however, use concept names defined in the background ontology. Thus the user DL can capture disjunction, if the background terminology provides a concept name for it. The methods proposed for this setting are tailored to the 'customization of background ontologies' approach for extending ontologies described in Section 1.2.2.

In this novel setting it is not clear whether the lcs exists or not and if it does, how it is computed. We investigate this inference for $\mathcal{ALC}$ as the background DL first for $\mathcal{EL}$ and then for $\mathcal{ALE}$ as the user DL.

In some cases it seems to be useful to relax the notion of the lcs to avoid over-fitting or for efficiency reasons. To address this issue, we describe practical approaches for computing *good* common subsumers to extend background terminologies, which may, however, not be the least ones.

# Chapter 4

# Concept approximation

In this chapter we investigate the inference 'concept approximation', which can be thought of as a service that translates concept descriptions from one DL to concept descriptions in another, typically less expressive DL. This inference is an instance of the general rewriting framework for DLs according to Definition 14 (on page 33) as introduced in [BKM00]. Concept approximation instantiates this framework in the following way: the source DL $\mathcal{L}_s$ and the TBox DL $\mathcal{L}_t$ are the same and the target DL $\mathcal{L}_d$ is a DL offering fewer concept constructors than $\mathcal{L}_s$. The binary relation $\rho$ between concept descriptions from source and target DL, as well as the ordering relation $\preceq$ between concept description from $\mathcal{L}_d$, is subsumption.

In the case we are interested in, the target DL does not offer concept disjunction, such that the obtained concept descriptions yield a meaningful generalization when computing their lcs (in the target DL). In principle one can compute an *upper approximation* by generalizing the input concept description or a *lower approximation* by specializing it. For our application we are only interested in *upper* approximation for two reasons. First, we approximate the concept description in order to obtain their generalization in a subsequent step, thus specializing them first is unintuitive. Second, and more importantly, there exist possibly many (minimal) specializations of one concept description, thus the choice of *one* specialization seems arbitrary. Moreover, it is not guaranteed to obtain a common subsumer at all, if the lcs is applied to lower approximations of the input concept descriptions. Thus we are only considering upper approximation and refer to it by the term approximation in the following.

The service of eliminating disjunctions from TBoxes and concept descriptions was already mentioned in [EBBK89; CBH92]. The motivation to consider this kind of service, was the already mentioned *knowledge base vivification*. Knowledge base vivification removes disjunctions form the TBox mainly to achieve better performance for reasoning tasks, but also for better understandability of the TBox for users with little expertise in logic.

In a similar way concept approximation can be employed to support user interfaces of DL systems. In the interaction with DL systems, users with little KR expertise may have difficulties to understand and make use of the full expressive power of the underlying DLs. To overcome this problem two approaches have been proposed in the literature. In [McG96] *concept pruning* is applied to achieve better readability of

the concept for naive users by removing too complex sub-concept descriptions. Now, instead of just eliminating sub-concept descriptions from the concept by removing them, computing their approximation would preserve as much information of the original concept description as possible and offer a way to display a simpler form of the original concept description.

The second approach for displaying concept descriptions in a more user-friendly way is to equip knowledge representation systems with a simplified *frame-based user interface* built on top of a more powerful DL system. Examples for such a systems are the TAMBIS system [BGB+99] and the ontology editors OilEd [BHGS01] or PROTÉGÉ [GMF+03]. On many occasions, these systems have to present concept descriptions to the user for editing, inspection, or as a solution of inference problems. Such concept descriptions, however, need not always fit into the restricted representation of the frame-based user interface or might be hard to comprehend for an inexperienced user. In such cases, approximation may be helpful as a means to represent concept descriptions in a simplified fashion suited to the user interface and the user's level of expertise.

Besides for computing simplified views on concept descriptions, concept approximation can be employed as a preprocessing step for other inferences. For example, new inferences such as concept matching, have been introduced to support the construction and maintenance of DL knowledge bases (see [Küs01; BT01b]). However, up to now they are mostly restricted to quite inexpressive DLs, for example to those that do not allow for concept disjunction. As in our case of the lcs, approximation can be used to overcome this problem to some extent by first approximating the concept descriptions and then applying the non-standard inferences to their approximations.

As we shall see, the computation algorithms for concept approximation are similar to the description-based algorithms for the lcs as discussed in the last chapter. These algorithms also require normalization of concept descriptions and structural characterization of subsumption as a basis. Unlike the case of the lcs, here these characterizations are given for the subsumption of a concept description in the source DL by a concept description in the target DL of concept approximation. Formally, the translation from source DL to target DL by concept approximation is defined as follows.

**Definition 30 (Approximation).** Let $C$ be an $\mathcal{L}_1$-concept description. A $\mathcal{L}_2$-concept description $D$ is an $\mathcal{L}_2$-*approximation* of $C$ iff

1. $C \sqsubseteq D$; and
2. $D \sqsubseteq E$ for every $\mathcal{L}_2$-concept description $E$ with $C \sqsubseteq E$.

$\diamond$

For the approximation-based approach to obtain meaningful least common subsumers, we are interested in the cases where $\mathcal{L}_1$ does offer disjunction, while $\mathcal{L}_2$ does not. Here, computing $\mathcal{L}_2$-approximations means to eliminate disjunctions from $\mathcal{L}_1$-concept descriptions while retaining as much information captured in the concept description as possible. In the simple case of a disjunction $C_1 \sqcup C_2$ of two $\mathcal{L}_2$-concepts, the most

specific $\mathcal{L}_2$-concept description subsuming $C_1 \sqcup C_2$ is just the lcs of $C_1$ and $C_2$. Hence, the disjunction is approximated by the lcs of the disjuncts. This observation holds in a more general setting and captures the close relationship between the lcs and the approximation inference.

**Corollary 31.** *Let $\mathcal{L}_2$ be a DL that does not allow to express disjunctions of arbitrary $\mathcal{L}_2$-concept descriptions, let $\mathcal{L}_1$ be the DL that extends $\mathcal{L}_2$ by disjunction and let $C_1, \ldots, C_n$ be $\mathcal{L}_2$-concept descriptions, then*

$$\mathsf{approx}_{\mathcal{L}_2}(\bigsqcup_{1 \leq i \leq n} C_i) \equiv \mathsf{lcs}_{\mathcal{L}_2}(C_1, \ldots, C_n).$$

Note that $\bigsqcup_{1 \leq i \leq n} C_i$ is a $\mathcal{L}_1$-concept description. The above equivalence follows directly from the definition of approximation and lcs. However, this equivalence is central for computation algorithms for approximations that eliminate disjunctions, since it allows to give computation algorithms more suitable for implementation as we will see.

In the remainder of this chapter we introduce the 'classic' computation algorithm for computing $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions as presented in [BKT01] or [Bra06]. Based on Corollary 31 we give a more efficient version of the algorithm and prove its correctness. Then we extend this computation algorithm by number restrictions to $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions and prove its correctness. To this end we give a structural characterization of subsumption of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions beforehand.

To be able to assess the information loss of a concept description by computing its approximation, we present the difference operator in Section 4.3. This operator computes a syntactic difference of two concept descriptions. We present a heuristic algorithm to compute the syntactic difference of an $\mathcal{ALC}$-concept description by an $\mathcal{ALE}$-concept description in this chapter in Section 4.3. This algorithm can be employed to assess the syntactic difference between an $\mathcal{ALC}$-concept description and its approximation.

## 4.1 Approximation of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions

The first computation algorithm for the non-standard inference approximation was devised by Brandt, Küsters and Turhan in [BKT02b; BKT01]. This work proposed a computation algorithm for approximations of $\mathcal{ALC}$-concept descriptions by $\mathcal{ALE}$-concept descriptions. We introduce this algorithm as a preparation for the method for concept approximation of $\mathcal{ALCN}$-concept descriptions by $\mathcal{ALEN}$-concept descriptions. For the proofs of soundness and completeness of the computation of $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions please refer to [BKT01] or [Bra06].

The method for computing $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions is again a structural method. A first idea to come up with a computation algorithm might be to replace each disjunction by the commonalities of its disjuncts. Proceeding in this way, an approximation algorithm traverses the syntax tree of the input

concept in a bottom-up fashion and substitutes a disjunction by the lcs of its disjuncts whenever one is found. This idea is formalized in the following definition of our first straightforward attempt to an approximation algorithm:

**Definition 32 (Approximation by substitution of disjunctions by their lcs).** The pseudo-approximation $\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C)$ of an $\mathcal{ALC}$ concept description $C$ by an $\mathcal{ALE}$ concept description is defined by:

$$\begin{aligned}
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C) &:= C \quad \text{, if } C \equiv \bot \text{ or } C \equiv \top \text{ or } C \in \mathit{prim}(C)\\
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_1 \sqcap \cdots \sqcap C_n) &:= \mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_1) \sqcap \cdots \sqcap \mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_n)\\
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_1 \sqcup \cdots \sqcup C_n) &:= \mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_1), \ldots, \mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_n)\}\big)\\
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(\exists r.C') &:= \exists r.\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C')\\
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(\forall r.C') &:= \forall r.\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C')
\end{aligned}$$

◇

Unfortunately, this naive approach does not always compute the most specific $\mathcal{ALE}$-concept description subsuming $C$. Let us consider the following example.

**Example 33 (Naive approximation).** *For atomic concepts $A$ and $B$, consider the $\mathcal{ALC}$-concept description $C_{ex} := (\forall r.B \sqcup (\exists r.B \sqcap \forall r.A)) \sqcap \exists r.A$.*

$$\begin{aligned}
\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_{ex}) &\equiv \mathsf{lcs}_{\mathcal{ALE}}(\forall r.B, \exists r.B \sqcap \forall r.A) \sqcap \exists r.A\\
&\equiv \forall r.\top \sqcap \exists r.A\\
&\equiv \exists r.A
\end{aligned}$$

*It is easy to verify that $C \sqsubseteq \exists r.(A \sqcap B) \sqsubset \mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}(C_{ex})$. Thus, the algorithm $\mathsf{c\text{-}approx}^{\mathrm{triv}}_{\mathcal{ALE}}$ did not find an optimal solution.*

The crucial point is to normalize full negation and nested con- and disjunctions before the propagation of value restrictions. If the approximation is to be computed w.r.t. a TBox and the concept descriptions contain names of defined concepts, the concept description has to be unfolded w.r.t. the TBox (according Definition 17) before normalization.

### 4.1.1 Normalization of $\mathcal{ALC}$-concept descriptions

In case of the DL $\mathcal{ALC}$ the normal form has to take care of all interactions of the $\mathcal{ALE}$-concept constructors in combination with full negation and disjunction. Negation is treated by transformation of the concept description into *negation normal form (NNF)*, which turns full negation into primitive negation by pushing all negations into the concept description until they occur only in front of concept names, using de Morgan' rules as well as the following rules.

**Definition 34 ($\mathcal{ALC}$-negation normal form).** An $\mathcal{ALC}$-concept description is in *$\mathcal{ALC}$-negation normal form* if the following rules have been applied exhaustively:

$$\begin{array}{rcl} \neg\neg C & \rightarrow & C \\ \neg\top & \rightarrow & \bot \\ \neg\bot & \rightarrow & \top \end{array} \qquad \begin{array}{rcl} \neg(C \sqcap D) & \rightarrow & (\neg C \sqcup \neg D) \\ \neg(C \sqcup D) & \rightarrow & (\neg C \sqcap \neg D) \\ \neg(\exists r.C) & \rightarrow & (\forall r.\neg C) \\ \neg(\forall r.C) & \rightarrow & (\exists r.\neg C). \end{array}$$

◇

Transforming an $\mathcal{ALC}$-concept description into NNF yields an equivalent concept description of the same size. To handle disjunctions correctly in our computation algorithm for approximation, we need to simplify (possibly) interleaved con- and disjunctions. We call a concept description *top-level* $\sqcup$*-free*, if it is in $\mathcal{ALC}$-negation normal form and does not contain any disjunction on the top role level. However, inside of existential or value restrictions of a top-level $\sqcup$-free concept descriptions disjunctions may appear. The following normal form transforms $\mathcal{ALC}$-concept descriptions into a disjunction of top-level $\sqcup$-free concept descriptions, i.e. the obtained concept description consists of (at most) one disjunction containing flattened conjunctions of (possibly negated) concept names, value and existential restrictions containing concept descriptions in $\mathcal{ALC}$-normal form.

**Definition 35 ($\mathcal{ALC}$-normal form).** Let $C$ be an $\mathcal{ALC}$-concept description. $C$ is in $\mathcal{ALC}$*-normal form*, iff $C = \bot$, $C = \top$, or $C$ is of the form

$$C = C_1 \sqcup \cdots \sqcup C_n \text{ and for all } i$$
$$C_i = \bigsqcap_{A \in prim(C_i)} A \sqcap \bigsqcap_{C' \in \mathsf{ex}_r(C_i)} \exists r.C' \sqcap \forall r.\mathsf{val}_r(C_i),$$

where (1) $C_i \not\equiv \bot$ for all $i$ and (2) $\mathsf{val}_r(C_i)$ and every concept in $\mathsf{ex}_r(C_i)$ again are in $\mathcal{ALC}$-normal form. ◇

Normalized $\mathcal{ALC}$-concept descriptions are in a similar form as disjunctive normal form on each role level. Obviously, $\mathcal{ALE}$-concept descriptions are always in $\mathcal{ALC}$-normal form. Note that every concept description in $\mathcal{ALC}$-normal form is also in NNF.

Transforming a concept description in $\mathcal{ALC}$-normal form can yield a concept description exponentially larger than the original in the worst case. For instance, computing the disjunctive normal form of $(A_1 \sqcup B_1) \sqcap \cdots \sqcap (A_n \sqcup B_n)$ produces a concept description of exponential size in $n$.

### 4.1.2 Computing $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions

A detailed discussion on the characterization of subsumption of an $\mathcal{ALC}$-concept description by an $\mathcal{ALE}$-concept description and proof of its correctness is given in [BKT01] and [Bra06]. We simply state the conditions for subsumption here, since we refer to them later.

**Theorem 36.** *Let $C = C_1 \sqcup \cdots \sqcup C_n$ be an $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form and $D$ an $\mathcal{ALE}$-concept description in $\mathcal{ALE}$-normal form. Then, $C \sqsubseteq D$ iff*

1. *$C \equiv \bot$ or $D \equiv \top$, or*

Input: $\mathcal{ALC}$-concept description $C$.
Output: $\mathcal{ALE}$-approximation $D$ of $C$.

1. If $C \equiv \bot$, then $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) := \bot$
2. If $C \equiv \top$, then $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) := \top$
3. Otherwise, transform $C$ into $\mathcal{ALC}$-normal form and return

$$\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) :=$$

$$\bigsqcap_{A \in \bigcap_i prim(C_i)} A$$

$$\sqcap \bigsqcap_{(C'_1,\ldots,C'_n) \in \mathsf{ex}_r(C_1) \times \cdots \times \mathsf{ex}_r(C_n)} \exists r.\mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C'_i \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big)$$

$$\sqcap \ \forall r.\mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big).$$

Figure 4.1: The algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$.

*2. for every $C_i$ with $i \in \{1, \ldots, n\}$ it holds that*

- *$prim(D) \subseteq prim(C_i)$, and*
- *$\forall D' \in \mathsf{ex}_r(D)\ \exists C' \in \mathsf{ex}_r(C_i) : C' \sqcap \mathsf{val}_r(C_i) \sqsubseteq D'$, and*
- *$\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(D)$.*

Intuitively, the conditions for subsumption are very similar to the conditions of the description-based characterization of subsumption between $\mathcal{ALE}$-concept descriptions from Theorem 24, besides the following two facts: here each disjunct from C is tested whether it fulfills the conditions and propagation of value restrictions onto existential restrictions is done 'on the fly'.

An algorithm to actually compute the $\mathcal{ALE}$-approximation of a given $\mathcal{ALC}$-concept description is displayed in Figure 4.1. Recall that we assume $N_R = \{r\}$. Obviously, the algorithm ensures that the input is transformed into $\mathcal{ALC}$-normal form. By computing $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C'_i \sqcap \mathsf{val}_r(C_i))$ instead of $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C'_i)$ for every existential restriction in the resulting concept description it is also ensured that value restrictions are propagated to existential restrictions. It should be noted that the argument $C'_i \sqcap \mathsf{val}_r(C_i)$ to $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ is not necessarily in $\mathcal{ALC}$-normal form—even if $C$ was transformed into $\mathcal{ALC}$-normal form before. To see how the above algorithm works, let us return to Example 33.

**Example 37.** *Again, consider the concept description $C_{ex} = (\forall r.B \sqcup (\exists r.B \sqcap \forall r.A)) \sqcap \exists r.A$. Applying $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ to $C_{ex}$ would firstly transform the input into $\mathcal{ALC}$-normal*

*form, yielding* $(\forall r.B \sqcap \exists r.A) \sqcup (\exists r.B \sqcap \forall r.A \sqcap \exists r.A)$. *According to the definition of* $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$, *we therefore have:*

$$\begin{aligned}
\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C_{ex}) &= \mathsf{c\text{-}approx}_{\mathcal{ALE}}((\forall r.B \sqcap \exists r.A) \sqcup (\exists r.B \sqcap \forall r.A \sqcap \exists r.A)) \\
&\equiv \exists r.\mathsf{lcs}_{\mathcal{ALE}}(A \sqcap B,\ B \sqcap A) \sqcap \\
&\quad\ \exists r.\mathsf{lcs}_{\mathcal{ALE}}(A \sqcap B,\ A \sqcap A) \sqcap \\
&\quad\ \forall r.\mathsf{lcs}_{\mathcal{ALE}}(B,\ A) \\
&\equiv \exists r.(B \sqcap A)\ \sqcap\ \exists r.A\ \sqcap \forall r.\top \\
&\equiv \exists r.(B \sqcap A).
\end{aligned}$$

This example shows that $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ correctly approximates the example concept. The following theorem states that the algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ always finds the correct approximation. The proof can be found in [BKT01; Bra06].

**Theorem 38.** *Let* $C$ *be an* $\mathcal{ALC}$*-concept description in* $\mathcal{ALC}$*-normal form. Then the concept description* $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C)$ *is the* $\mathcal{ALE}$*-approximation of* $C$, *i.e.,*

1. $C \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALE}}(C)$, *and*
2. $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) \sqsubseteq D$ *for every* $\mathcal{ALE}$*-concept description* $D$ *with* $C \sqsubseteq D$.

As each $\mathcal{ALC}$-concept description can be transformed into an equivalent $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form we may extend the above result in the following way:

**Corollary 39.**
1. *The above result also holds for* $\mathcal{ALC}$*-concept descriptions which are not in* $\mathcal{ALC}$*-normal form.*
2. *The size of* $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C)$ *can be exponential in the size of* $C$, *where* $C$ *is in* $\mathcal{ALC}$*-normal form.*

**Proof.** 1. Easy to see since (1) the algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ starts by computing the $\mathcal{ALC}$-normal form of its input and (2) $\top$ and $\bot$ are represented uniquely in $\mathcal{ALC}$-normal form.

2. Consider two $\mathcal{ALE}$-concept descriptions $C_1$ and $C_2$ in $\mathcal{ALE}$-normal form. According to the definition, $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C_1 \sqcup C_2) = \mathsf{lcs}_{\mathcal{ALE}}(C_1, C_2)$. It has been shown in [BKM99] that there exist pairs of $\mathcal{ALE}$-concept descriptions whose lcs is exponentially large in the size of the input. ❑

A natural question regards the computational complexity of the approximation algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$. Since the algorithm uses the computation algorithm for the lcs in $\mathcal{ALE}$ as a sub-procedure, we know that computation of $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions is necessarily worst case exponential. The following Proposition gives an upper bound. It is shown in [BKT01; Bra06] that $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ can be realized as a double-exponential time algorithm.

**Proposition 40.** *The algorithm* $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ *is a* 2-ExpTime *algorithm, i.e., for a given* $\mathcal{ALC}$*-concept description the computation of* $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C)$ *takes at most double exponential time in the size of* $C$.

We give an intuition for this result. The main sources of complexity are the computation of the $\mathcal{ALC}$-normal form and the number of recursive calls for the computation of existential restrictions. The computation of the $\mathcal{ALC}$-normal form yields for a concept $C$ with $|C| = n$ a concept description with exponentially many ($2^{p(n)}$ for some polynomial $p$) disjuncts on the top-level each of which is limited in size by $n$. The normalized concept description has exponentially many disjuncts $C_i$ with a linear number of existential restrictions $C_i'$. The number of existential restrictions to be computed in

$$\prod_{(C_1',\ldots,C_n') \in \mathsf{ex}_r(C_1) \times \cdots \times \mathsf{ex}_r(C_n)} \exists r.\mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}\big(C_i' \sqcap \mathsf{val}_r(C_i)\big) \mid 1 \leq i \leq n\}\big)$$

is double exponential in $n$. For every such existential restriction a lcs of a set of exponential cardinality must be computed. Thus, the computation tree of $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$, is of double exponential size in the size of $C$. To sum up, computing approximations of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions is worst case exponential and in 2-ExpTime. Whether or not there also exists an exponential time approximation algorithm is an open problem.[16]

The algorithm for computing $\mathcal{ALE}$ approximations of $\mathcal{ALC}$-concept descriptions can also be used to approximate concept descriptions in less expressive DLs than $\mathcal{ALE}$. To this end, one removes the sub-concept descriptions in the $\mathcal{ALE}$-approximation that use concept constructors which are not supported by the smaller target DL. For example, we can obtain an $\mathcal{EL}$ approximation of an $\mathcal{ALC}$-concept description, if we remove the value restrictions from the concept description from the approximation obtained by the algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$.

## 4.2 Approximation of $\mathcal{ALCN}$- by $\mathcal{ALEN}$-concept descriptions

To be able to obtain meaningful least common subsumers also in the presence of number restrictions, we extend the method for computing approximations introduced in the last section. The method described in the following was published in [BKT02a]. However, the proof of its correctness was omitted there and is now supplied in this Section. We have to start by computing the induced information captured in $\mathcal{ALCN}$-concept descriptions. This requires a combination of $\mathcal{ALC}$-normal form and the methods for extracting induced information from $\mathcal{ALEN}$-concept descriptions presented in Section 3.3.1.

### 4.2.1 Induced information in $\mathcal{ALCN}$-concept descriptions

In order to compute $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions, one needs to pre-process the $\mathcal{ALCN}$-concept descriptions, i.e., push negation inward, turn concepts

[16] In [DT02] an attempt has been made to propose a deterministic exponential time algorithm to compute $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions. However, in [DT03] the same authors claim that even w.r.t. a certain more compact representation of concept descriptions, no deterministic exponential time algorithm computing $\mathcal{ALE}$-approximations exists.

into a kind of disjunctive normal form and make implicit facts explicit, i.e., compute induced concepts. To treat full negation, we have to transform concept descriptions into $\mathcal{ALCN}$-negation normal form.

**Definition 41 ($\mathcal{ALCN}$-negation normal form).** An $\mathcal{ALCN}$-concept description is in *$\mathcal{ALCN}$-negation normal form* if the rules for $\mathcal{ALC}$-negation normal form (Definition. 34 on page 64) and the following rules have been applied exhaustively:

$$\neg(\leq n\ r) \quad \rightarrow \quad (\geq (n+1)\ r) \tag{4.1}$$

$$\neg(\geq n\ r) \quad \rightarrow \quad \begin{cases} (\leq 0\ r), & \text{if } n = 0 \\ (\leq (n-1)\ r), & \text{otherwise} \end{cases} \tag{4.2}$$

$\diamond$

This normal form obviously preserves equivalence of $\mathcal{ALCN}$-concept descriptions and does not increase the concept size. Next, we define the normal form that transforms $\mathcal{ALCN}$-concept descriptions into a disjunction of top-level $\sqcup$-free concept descriptions (on each role level). This normal form uses the notions of induced number restrictions $\mathsf{min}_r$ and $\mathsf{max}_r$ as defined in Section 3.3.1.

**Definition 42 ($\mathcal{ALCN}$-normal form).** Let $C$ be an $\mathcal{ALCN}$-concept description. $C$ is in *$\mathcal{ALCN}$-normal form* iff $C = \bot$, $C = \top$, or $C$ is of the form $C = C_1 \sqcup \ldots \sqcup C_n$ with $C_i :=$

$$\bigsqcap_{A \in prim(C_i)} A \ \sqcap \bigsqcap_{C' \in \mathsf{ex}_r(C_i)} \exists r.C' \ \sqcap \ \forall r.\mathsf{val}_r(C_i) \ \sqcap \ (\geq \mathsf{min}_r(C_i)\ r) \ \sqcap \ (\leq \mathsf{max}_r(C_i)\ r),$$

where (1) $C_i \not\equiv \bot$ for all $i$ and (2) $\mathsf{val}_r(C_i)$ and every concept in $\mathsf{ex}_r(C_i)$ again are in $\mathcal{ALCN}$-normal form. $\diamond$

It is easy to see that each $\mathcal{ALCN}$-concept description has an equivalent concept description in $\mathcal{ALCN}$-normal form. As in case of $\mathcal{ALC}$-concept descriptions, the resulting normal form may be of size exponential in the size of the given concept description, due to the distribution of conjuncts over disjuncts.

For $\mathcal{ALC}$, the computation of induced information is easy from a conceptual point of view, since it suffices to make inconsistencies explicit and propagate value restrictions to existential restrictions in each disjunct. In case of $\mathcal{ALCN}$ number restrictions can induce new value and existential restrictions. To extract the information implicitly captured in $\mathcal{ALCN}$-concept descriptions in $\mathcal{ALCN}$-normal form, one has to apply to each disjunct the same methods to compute induced information as for $\mathcal{ALEN}$-concept descriptions as discussed in Section 3.3. Please note that a disjunct of a concept description in $\mathcal{ALCN}$-normal form is top-level $\sqcup$-free and thus only uses $\mathcal{ALEN}$-concept constructors on the top most role level. We can therefore use the notions for existential mappings ($\alpha()$), the set of merged concept descriptions ($\mathsf{ex}_r()^\alpha$) and the set of all mappings ($\Gamma_r()$) directly for disjuncts of a normalized $\mathcal{ALCN}$-concept description as they were introduced in Section 3.3.1.

Some of the methods for computing induced information of $\mathcal{ALEN}$-concept descriptions use the lcs in $\mathcal{ALEN}$ to express the commonalities of the different existential

mappings in $\mathcal{ALEN}$. Since in $\mathcal{ALCN}$ concept disjunction is available, one can easily express the commonalities of the existential mappings simply by taking the disjunction of the respective concept descriptions.[17] In what follows, let $C = C_1 \sqcup \cdots \sqcup C_n$ be in $\mathcal{ALCN}$-normal form. The induced number restrictions of $C$ are $(\geq \mathsf{min}_r(C)\, r)$, $(\leq \mathsf{max}_r(C)\, r)$ and the set of all merged existential restrictions is $\mathsf{ex}_r()^*$ as defined in Section 3.3.1.

**Induced existential restrictions.** We use the notions for concept mappings as introduced in Section 3.3.1. The set of induced existential restrictions $\mathsf{ind\text{-}ex}_r(C_i)$ of a top-level $\sqcup$-free $C_i$ is then defined by means of a given set of $k_i$ concept mapping functions $\Gamma_{ir}(C_i) = \{\alpha_{i1}, \ldots, \alpha_{ik_i}\}$ and $\mathsf{ind\text{-}ex}_r(C_i)$ as follows:

**Definition 43 (Induced existential restrictions in $\mathcal{ALCN}$).** Let $C = C_1 \sqcup \cdots \sqcup C_n$ be an $\mathcal{ALCN}$-concept description in $\mathcal{ALCN}$-normal form and for each $C_1$ $(1 \leq i \leq n)$ let $\Gamma_{ir}(C_i) = \{\alpha_{i1}, \ldots, \alpha_{ik_i}\}$ the set of all existential mappings for $C_1$. Then $\mathsf{ind\text{-}ex}_r(C_i)$ is defined as:

1. if $\mathsf{ex}_r(C_i) \neq \emptyset$, then

$$\begin{aligned}\mathsf{ind\text{-}ex}_r(C_i) \quad &:= \quad \big\{\, \mathsf{lcs}_{\mathcal{ALCN}}\big(\{(C_{il} \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq l \leq k_i\}\big) \\ &\qquad\qquad \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \big\}; \\ &\equiv \quad \big\{\, \textstyle\bigsqcup_{1 \leq l \leq k_i}(C_{il} \sqcap \mathsf{val}_r(C_i)) \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \big\};\end{aligned}$$

2. if $\mathsf{ex}_r(C_i) = \emptyset$ and $\mathsf{min}_r(C_i) \geq 1$, then $\mathsf{ind\text{-}ex}_r(C_i) := \{\mathsf{val}_r(C_i)\}$;

3. otherwise, $\mathsf{ind\text{-}ex}_r(C_i) := \emptyset$.

$\diamond$

The value restrictions have already been propagated onto the respective existential restrictions in $\mathsf{ind\text{-}ex}_r(C_i)$. Note that the concept descriptions obtained in the first case are not necessarily in $\mathcal{ALCN}$-normal form (even if the disjunction is flattened).

**Induced value restrictions.** Recall that new value restrictions can only be induced for two reasons: by $\mathsf{max}_r(C) = 0$ or by the merging of existential restrictions in combination with at-most restrictions. In the case of $\mathcal{ALCN}$-concept descriptions we have to check this for each disjunct $C_i$ of $C$. With all merged existential restrictions collected in $\mathsf{ex}_r(C_i)^* = \bigcup_{\alpha_i \in \Gamma_{ir}(C_i)} \mathsf{ex}_r(C_i)^{\alpha_i}$, the induced value restriction $\mathsf{ind\text{-}val}_r(C_i)$ of $C_i$ is defined as follows:

**Definition 44 (Induced value restrictions in $\mathcal{ALCN}$).** Let $C = C_1 \sqcup \cdots \sqcup C_n$ be an $\mathcal{ALCN}$-concept description in $\mathcal{ALCN}$-normal form, then $\mathsf{ind\text{-}val}_r(C_i)$ is defined as:

[17] In [BKT02a] was a slightly different approach pursued, by first computing the $\mathcal{ALEN}$-approximation of each of the concepts obtained from existential mappings and then computing their lcs in $\mathcal{ALEN}$. Our approach here is easier in conceptual and practical respects, while it yields the same result. Conceptually, it is easier since it simply uses the language features available in $\mathcal{ALCN}$ and practically, since it avoids unnecessary computation of the $\mathcal{ALEN}$-lcs and -approximation as a preprocessing step to $\mathcal{ALEN}$-approximation.

1. if $\mathsf{max}_r(C_i) = 0$, then $\mathsf{ind\text{-}val}_r(C_i) := \bot$;

2. if $0 < \kappa_r(C_i) < \mathsf{max}_r(C_i)$, then $\mathsf{ind\text{-}val}_r(C_i) := \mathsf{val}_r(C_i)$;

3. if $0 < \kappa_r(C_i) = \mathsf{max}_r(C_i)$, then
$\mathsf{ind\text{-}val}_r(C_i) := \mathsf{lcs}_{\mathcal{ALCN}}(\{(\mathsf{val}_r(C_i) \sqcap C') \mid C' \in \mathsf{ex}_r(C_i)^*\})$
$\equiv \bigsqcup_{C' \in \mathsf{ex}_r(C_i)^*}(\mathsf{val}_r(C_i) \sqcap C')$

$\diamond$

Again, the concept description obtained by the lcs does not have to be in $\mathcal{ALCN}$-normal form. Equipped with a method for extracting implicit information from $\mathcal{ALCN}$-concept descriptions, we now give a characterization of subsumption.

### 4.2.2 Structural characterization of subsumption

We provide a structural characterization of subsumption of $\mathcal{ALCN}$-concept descriptions subsumed by $\mathcal{ALEN}$-concept descriptions. This characterization is later employed to prove correctness of our approximation algorithm. Assume $C$ is an $\mathcal{ALCN}$-concept description in $\mathcal{ALCN}$-normal form. It is easy to see that $C$ is subsumed by an $\mathcal{ALEN}$-concept description $D$ if and only if every disjunct $C_i$ in $C$ is subsumed by $D$. The following theorem extends the characterization of subsumption of two $\mathcal{ALEN}$-concept descriptions by treatment of disjunction to a method for $\mathcal{ALCN}$-concept descriptions. The characterization of subsumption for $\mathcal{ALEN}$-concept descriptions stated in Theorem 26 (on page 53) was published and proven correct in [KM01b].

**Theorem 45.** *Let $C$ be an $\mathcal{ALCN}$-concept description in $\mathcal{ALCN}$-normal form with $n$ disjuncts $C_1, \ldots, C_n$ and let $D$ be an $\mathcal{ALEN}$-concept description in $\mathcal{ALEN}$-normal form. Then, $C \sqsubseteq D$ iff $C \equiv \bot$, $D \equiv \top$, or for every $i \in \{1, \ldots, n\}$ it holds that*

1. *$prim(D) \subseteq prim(C_i)$, and*
2. *$\mathsf{max}_r(C_i) \leq \mathsf{max}_r(D)$, and*
3. *$\mathsf{min}_r(C_i) \geq \mathsf{min}_r(D)$, and*
4. *for every $D' \in \mathsf{ex}_r(D)$, there exists an existential restriction in $C' \in \mathsf{ind\text{-}ex}_r(C_i)$ s.t. $C' \sqcap \mathsf{val}_r(C_i) \sqsubseteq D'$, and*
5. *if $\mathsf{val}_r(D) \not\equiv \top$, then $\mathsf{ind\text{-}val}_r(C_i) \sqsubseteq \mathsf{val}_r(D)$.*

**Proof.** ($\Rightarrow$) If $C \equiv \bot$ or $D \equiv \top$ nothing has to be shown. Assume $\bot \sqsubset C \sqsubseteq D \sqsubset \top$. By definition of $\mathcal{ALCN}$-normal form, $C_i$ is consistent. We have to show that the Conditions 1–5 hold for all $C_i$.

1. Assume $prim(D) \not\subseteq prim(C_i)$ for one $i$. Then there exists an $A \in prim(D) \setminus prim(C_i)$. By definition of the $\mathcal{ALCN}$-normal form, $C_i$ is consistent. We may therefore consider a canonical interpretation $\mathcal{I}$ of $C_i$. By definition, the individual $d \in \Delta^{\mathcal{I}}$ for $C_i$ does not occur in $A^{\mathcal{I}}$, since $A \notin prim(C_i)$. Thus, $d \notin D^{\mathcal{I}}$ and therefore $C \not\sqsubseteq D$, in contradiction to our assumption.

2. Assume $\mathsf{max}_r(C_i) > \mathsf{max}_r(D)$ for one $i$. By the definition of $\mathsf{max}_r(C_i)$ there exists a model $\mathcal{I}$ and $d \in \Delta$ with $d \in C_i^{\mathcal{I}}$ s.t. $d$ has $\mathsf{max}_r(D)+1$ $r$-successors; otherwise we would have $\mathsf{max}_r(C_i) \leq \mathsf{max}_r(D)$. Obviously, $d \notin D^{\mathcal{I}}$. Thus, $\mathsf{max}_r(C_i) \leq \mathsf{max}_r(D)$.

3. Assume $\mathsf{min}_r(C_i) < \mathsf{min}_r(D)$ for one $i$. Since $C_i$ is satisfiable, there exists a tree model $\mathcal{I}$ with root $d$ of $C_i$ s.t. $d \in C_i^{\mathcal{I}}$ and $d$ has exactly $\mathsf{min}_r(C_i)$ different $r$-successors in $\mathcal{I}$. Obviously, $d \notin D^{\mathcal{I}}$. Thus, $\mathsf{min}_r(C_i) \geq \mathsf{min}_r(D)$.

4. Let $D' \in \mathsf{ex}_r(D)$. We have to distinguish two cases for $\mathsf{ind\text{-}ex}_r(C_i)$.

- Assume $\mathsf{ex}_r(C_i) = \emptyset$. If $\mathsf{min}_r(C_i) = 0$, then there exists a tree model $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ with root $x_0$ of $C$ with $d \in C_i^{\mathcal{I}}$ and $d$ has 0 $r$-successors in $\mathcal{I}$. Consequently, $d \notin D^{\mathcal{I}}$ in contradiction to $C \sqsubseteq D$ and $C_i \sqsubseteq D$. Hence, $\mathsf{min}_r(C_i) \geq 1$.

  Assume $\mathsf{val}_r(C_i) \not\sqsubseteq D'$, i.e., there exists a tree model $\mathcal{I}' = (\Delta_{\mathcal{I}'}, \cdot^{\mathcal{I}})$ with root $y_0$ s.t. $y_0 \in (\mathsf{val}_r(C_i) \sqcap \bigsqcap_{A \in prim(C_i)} A)^{\mathcal{I}'}$ and $y_0 \notin D'^{\mathcal{I}'}$. Let $\mathcal{I}_j, 1 \leq j \leq \mathsf{min}_r(C_i)$, be pairwise disjoint copies of $\mathcal{I}'$ with roots $y_i$, which are pairwise disjoint with $\mathcal{I}$. Define $\mathcal{J}' = (\Delta_{\mathcal{J}'}, \cdot^{\mathcal{J}})$ as :

  - $\Delta_{\mathcal{J}} := \Delta_{\mathcal{I}} \cup \bigcup_{1 \leq j \leq \mathsf{min}_r(C_i)} \Delta_{\mathcal{I}_j}$;
  - $P^{\mathcal{J}} := P^{\mathcal{I}} \cup \bigcup_{1 \leq j \leq \mathsf{min}_r(C_i)} P^{\mathcal{I}_j}$ for all $P \in N_C$;
  - $r^{\mathcal{J}} := \bigcup_{1 \leq j \leq \mathsf{min}_r(C_i)} r^{\mathcal{I}_j} \ \cup \ \big\{(x_0, y_j) \mid 1 \leq j \leq \mathsf{min}_r(C_i)\big\}$.

  It is easy to see that $\mathcal{J}$ is a model of $C$ with $x_0 \in C^{\mathcal{I}}$, but since there does not exist an $r$-successor $y$ of $x_0$ with $y \in D'^{\mathcal{J}}$, it is $x_0 \notin D^{\mathcal{I}}$ in contradiction to $C \sqsubseteq D$. Thus $\mathsf{val}_r(C) \sqsubseteq D'$.

- Now assume $\mathsf{ex}_r(C_i) \neq \emptyset$. We have to show that there exists a concept $C' \in \big\{ \bigsqcup_{1 \leq l \leq k_i} (C_{il} \sqcap \mathsf{val}_r(C_i)) \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \big\}$ with $C' \sqsubseteq D'$. To show this, we must prove that for each $\alpha_{ij} \in \Gamma_{ir}(C_i)$ there exists $C'' \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}$ with $C'' \sqcap \mathsf{val}_r(C_i) \sqsubseteq D'$. Assume that there exists an $\alpha' \in \Gamma_{ir}(C_i)$ s.t. $C'' \sqcap \mathsf{val}_r(C_i) \not\sqsubseteq D'$ for all $C'' \in \mathsf{ex}_r(C_i)^{\alpha'}$. Let $C_{\alpha'}$ be the concept description obtained from $C_i$ by removing all existential restrictions from the top-level of $C_i$ and conjoining all existential restrictions from $\{\exists r.C'' \mid C'' \in \mathsf{ex}_r(C_i)^{\alpha'}\}$. We define a tree model $\mathcal{I}$ with root $x_0$ s.t. $x_0 \in C_{\alpha'}^{\mathcal{I}} \subseteq C_i^{\mathcal{I}}$ and $x_0 \notin D^{\mathcal{I}}$ as follows:

  - Define $\mathcal{I}_0 := (\{x_0\}, \cdot^{\mathcal{I}_0})$ with
    - $P^{\mathcal{I}_0} := \begin{cases} \{x_0\}, & P \in prim(C_i); \\ \emptyset, & P \notin prim(C_i) \end{cases}$ for all $P \in N_C$;
    - $r^{\mathcal{I}_0} := \emptyset$.
  - For each $C'' \in \mathsf{ex}_r(C_i)^{\alpha'}$, let $\mathcal{I}_{C''} := (\Delta_{\mathcal{I}_{C''}}, \cdot^{\mathcal{I}_{C''}})$ be a tree model with root $y_{C''}$ s.t. $y_{C''} \in (C'' \sqcap \mathsf{val}_r(C_i))^{\mathcal{I}_{C''}}$ and $y_{C''} \notin D'^{\mathcal{I}_{C''}}$
  - If $\mathsf{min}_r(C_i) > |\mathsf{ex}_r(C_i)|$, then let $\mathcal{I}_1, \ldots, \mathcal{I}_l$ with $l := \mathsf{min}_r(C_i) - |\mathsf{ex}_r(C_i)|$, be copies of $\mathcal{I}_{C''}$, as defined above, for some $C''$ with root $y_j, 1 \leq j \leq l$.

  W.l.o.g. let all these interpretations be pairwise disjoint. Now, define $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ by:

- $\Delta_{\mathcal{I}} := \{x_0\} \cup \bigcup_{1\leq i\leq l} \Delta_{\mathcal{I}_i} \cup \bigcup_{C''\in \mathsf{ex}_r(C_i)^{\alpha'}} \Delta_{\mathcal{I}_{C''}}$;
- $P^{\mathcal{I}} := P^{\mathcal{I}_0} \cup \bigcup_{1\leq i\leq l} P^{\mathcal{I}_i} \cup \bigcup_{C''\in \mathsf{ex}_r(C_i)^{\alpha'}} P^{\mathcal{I}_{C''}}$ for all $P \in N_C$;
- $r^{\mathcal{I}} := \{(x_0, y_{C''}) \mid C'' \in \mathsf{ex}_r(C)^{\alpha'}\} \cup \{(x_0, y_i) \mid 1 \leq i \leq l\} \cup \bigcup_{1\leq i\leq l} r^{\mathcal{I}_i} \cup \bigcup_{C''\in \mathsf{ex}_r(C_i)^{\alpha'}} r^{\mathcal{I}_{C''}}$.

By construction, $\mathcal{I}$ is a model of $C_{\alpha'}$ and hence of $C_i$ with $x_0 \in C^{\mathcal{I}}$. But since there is no $r$-successor $y$ of $x_0$ with $y \in D'^{\mathcal{I}}$, $x_0 \notin D^{\mathcal{I}}$ follows in contradiction to $C_i \sqsubseteq D$ and thus $C \sqsubseteq D$. Thus for each $\alpha_i \in \Gamma_{ir}(C_i)$ there exists $C'' \in \mathsf{ex}_r(C_i)^{\alpha_i}$ with $C' \sqcap \mathsf{val}_r(C_i)$. Thus there exists a concept $C' \in \left\{ \bigsqcup_{1\leq l\leq k_i}(C_{il} \sqcap \mathsf{val}_r(C_i)) \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \right\}$ with $C' \sqsubseteq D'$.

5. Assume $\mathsf{ind\text{-}val}_r(C_i) \not\sqsubseteq \mathsf{val}_r(D)$ for one $i$. Assume $\mathsf{val}_r(D) \not\equiv \top$ and $\mathsf{max}_r(C_i) \not\equiv \infty$. Otherwise nothing has to be shown. We have to distinguish two cases:

1. Case: $\kappa_r(C_i) < \mathsf{max}_r(C_i)$. Since in this case $\mathsf{ind\text{-}val}_r(C_i) = \mathsf{val}_r(C_i)$, we have to show $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(D)$. Proof by contraposition: assume $\mathsf{val}_r(C_i) \not\sqsubseteq \mathsf{val}_r(D)$. We define a tree model $\mathcal{I}$ with root $x_0$ of $C_i$ s.t. $x_0 \in C_i^{\mathcal{I}}$ and $x_0 \notin D^{\mathcal{I}}$. In order to define $\mathcal{I}$, we have to distinguish two cases:

   (a) there exists $(\leq m\ r)$ on top-level of $C_i$ with $m > \kappa_r(C_i)$, and

   (b) there exists no such number restriction.

   Case (a): The $\mathcal{ALCN}$-concept description $C_i'$ is obtained from $C_i$ by removing $(\leq m\ r)$ from the top-level of $C_i$. Obviously, is $\kappa_r(C_i') = \kappa_r(C_i)$. Let $\mathcal{I}'$ be a tree model of $C_i'$ with root $x_0$ s.t. $x_0 \in C_i'^{\mathcal{I}'}$ and $x_0$ has exactly $\kappa_r(C_i')$ $r$-successors in $\mathcal{I}'$. By assumption there exists a model $\mathcal{I}''$ of $\mathsf{val}_r(C_i)$ with root $y_0$ s.t. $y_0 \in \mathsf{val}_r(C_i)^{\mathcal{I}''}$ and $y_0 \notin \mathsf{val}_r(D)^{\mathcal{I}''}$. Let $l = m - \kappa_r(C_i)$ and let $\mathcal{I}_i$, with $1 \leq i \leq l$, be disjoint copies of $\mathcal{I}''$ with roots $y_i$, which are also pairwise disjoint with $\mathcal{I}'$. Define $\mathcal{I} := (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ with

   - $\Delta_{\mathcal{I}} := \Delta'_{\mathcal{I}} \cup \bigcup_{1\leq i\leq l} \Delta_{\mathcal{I}_i}$;
   - $P^{\mathcal{I}} := P^{\mathcal{I}'} \cup \bigcup_{1\leq i\leq l} P^{\mathcal{I}_i}$ for all $P \in N_C$;
   - $r^{\mathcal{I}} := r^{\mathcal{I}'} \cup \bigcup_{1\leq i\leq l} r^{\mathcal{I}_i} \cup \{(x_0, y_i) \mid 1 \leq i \leq l)\}$.

   Obviously, $\mathcal{I}$ is a model of $C_i$ with $x_0 \in C_i^{\mathcal{I}}$, but since $x_0$ has at least the $r$-successor $y_1$ with $y_1 \notin \mathsf{val}_r(D)^{\mathcal{I}}$, we get a contradiction to $C_i \sqsubseteq D$. Thus, $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(D)$.

   Case (b): Let $\mathcal{I}'$ be a tree model of $C_i'$ with root $x_0$ s.t. $x_0 \in C_i^{\mathcal{I}}$ and $x_0$ has exactly $\kappa_r(C_i)$ $r$-successors in $\mathcal{I}'$. By assumption, there exists a model $\mathcal{I}''$ of $\mathsf{val}_r(C_i)$ with root $y_0$ s.t. $y_0 \in \mathsf{val}_r(C_i)^{\mathcal{I}''}$ and $y_0 \notin \mathsf{val}_r(D)^{\mathcal{I}''}$. Define $I := (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ with

   - $\Delta_{\mathcal{I}} := \Delta_{\mathcal{I}'} \cup \Delta_{\mathcal{I}''}$;
   - $P^{\mathcal{I}} := P^{\mathcal{I}'} \cup P^{\mathcal{I}''}$ for all $P \in N_C$;

- $r^{\mathcal{I}} := r^{\mathcal{I}'} \cup r^{\mathcal{I}''} \cup \{(x_0, y_0)\}$.

Due to $\kappa_r(C_i) < \mathsf{max}_r(C_i)$, $\mathcal{I}$ is a model of $C_i$ with $x_0 \in C_i^{\mathcal{I}}$, but since $x_0$ has at least the $r$-successor $y_0$ with $y_0 \notin \mathsf{val}_r(D)^{\mathcal{I}}$, we get $x_0 \notin D^{\mathcal{I}}$ in contradiction to $C_i \sqsubseteq D$. Thus $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(D)$.

2. Case: $\kappa_r(C_i) = \mathsf{max}_r(C_i)$. Thus $\mathsf{ind\text{-}val}_r(C_i) = \bigsqcup_{C' \in \mathsf{ex}_r(C_i)^*}(\mathsf{val}_r(C_i) \sqcap C')$. Assume there exists $C'' \in \mathsf{ex}_r(C_i)^*$ with $\mathsf{val}_r(C_i) \sqcap C'' \not\sqsubseteq \mathsf{val}_r(D)$. Then there exists $\alpha_i' \in \Gamma_{ir}(C_i)$ s.t. $C'' \in \mathsf{ex}_r(C_i)^{\alpha_i'}$. Let $C_{\alpha'}$ be the $\mathcal{ALCN}$-concept description obtained from $C_i$ by exchanging all existential restrictions by the ones from $\mathsf{ex}_r(C_i)^{\alpha_i'}$, thus $C_{\alpha_i'} \sqsubseteq C_i$. Since the existential mappings have to be consistent, $C_{\alpha_i'} \not\equiv \bot$, there exists a tree model $\mathcal{I}$ of $C_{\alpha_i'}$ with root $x_0 \in C_{\alpha_i'}^{\mathcal{I}}$. Since $\mathsf{max}_r(C_i) = \kappa_r(C_i)$, for every $\widehat{C} \in \mathsf{ex}_r(C_i)^{\alpha_i'}$, there exists exactly one $r$-successor $y$ of $x_0$ in $\mathcal{I}$ with $y \in \widehat{C}^{\mathcal{I}}$, $x_0$ has no other $r$-successors than $y$ with $y \in (\mathsf{val}_r(C_i) \sqcap C'')^{\mathcal{I}}$. By assumption there exists a tree model $\mathcal{I}' = (\Delta_{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ with root $y_0$ s.t. $y_0 \in (\mathsf{val}_r(C) \sqcap C'')^{\mathcal{I}'}$. Define a new model $\mathcal{J} = (\Delta_{\mathcal{J}}, \cdot^{\mathcal{J}})$ with

   - $\Delta_{\mathcal{J}} := \Delta_{\mathcal{I}} \cup \Delta_{\mathcal{I}'}$;
   - $P^{\mathcal{J}} := P^{\mathcal{I}} \cup P^{\mathcal{I}'}$ for all $P \in N_C$;
   - $r^{\mathcal{J}} := r^{\mathcal{I}'} \setminus \{(x_0, y)\} \cup r^{\mathcal{I}'} \cup \{(x_0, y_0)\}$.

   Then $\mathcal{J}$ is a model of $C_i$ with $x_0 \in C_i^{\mathcal{J}}$. But since the $r$-successor $y_0$ of $x_0$ in $\mathcal{J}$ is not an instance of $\mathsf{val}_r(D)^{\mathcal{J}}$, $x_0 \notin D^{\mathcal{J}}$ follows in contradiction to $C_i \sqsubseteq D$ and $C \sqsubseteq D$. Thus, $\mathsf{val}_r(C_i) \sqcap C'' \sqsubseteq \mathsf{val}_r(D)$ for all $C'' \in \mathsf{ex}_r(C_i)^*$.

($\Leftarrow$): If $C \equiv \bot$ or $D \equiv \top$, then $C \sqsubseteq D$. Assume $C \not\equiv \bot$ and $D \not\equiv \top$, and Conditions 1 to 5 are satisfied for $C$. It suffices to show $C_i \sqsubseteq D$. We show that for any model $\mathcal{I}$ of $C_i$ with $x \in C_i^{\mathcal{I}}$ also $x \in D^{\mathcal{I}}$ holds. It is sufficient to show $x \in D'^{\mathcal{I}}$ for each conjunct $D'$ occurring on the top-level of $D$ since $D$ is in $\mathcal{ALEN}$-normal form.

- Let $D' \in \mathit{prim}(D)$: Since $\mathit{prim}(D) \subseteq \mathit{prim}(C_i)$ holds and $x \in C_i^{\mathcal{I}}$, we obtain $x \in D'^{\mathcal{I}}$.

- Let $D' = (\leq\ n\ r)$: By Condition 2, it holds that $\mathsf{max}_r(C_i) \leq \mathsf{max}_r(D) \leq n$. Since $x \in C_i^{\mathcal{I}}$, it follows that $x \in (\leq\ \mathsf{max}_r(C_i)\ r)^{\mathcal{I}} \subseteq (\leq\ n\ r)^{\mathcal{I}}$.

- Let $D' = (\geq n\ r)$: By Condition 3, it holds that $\mathsf{min}_r(C_i) \geq \mathsf{min}_r(D) \geq n$. Since $x \in C_i^{\mathcal{I}}$, it follows that $x \in (\geq\ \mathsf{min}_r(C_i)\ r)^{\mathcal{I}} \subseteq (\geq\ n\ r)^{\mathcal{I}}$.

- Let $D' = \exists r.D''$: we have to show there exists an existential restriction in $C_i' \in \mathsf{ind\text{-}ex}_r(C_i)$ s.t. $(C_i' \sqcap \mathsf{val}_r(C_i))^{\mathcal{I}} \subseteq D'^{\mathcal{I}}$. Since $D' = \exists r.D''$ implies $\mathsf{min}_r(D') \geq 1$ and Condition 3 holds, we have $\mathsf{min}_r(C_i) \geq 1$. We distinguish between the two remaining cases from the definition of $\mathsf{ind\text{-}ex}_r()$:

  - $\mathsf{ex}_r(C_i) = \emptyset$. Thus $\mathsf{ind\text{-}ex}_r(C_i) = \mathsf{val}_r(C_i)$, consequently by assumption $\mathsf{val}_r(C_i) \sqsubseteq D''$. Hence, with $\mathsf{min}_r(C_i) \geq 1$ there exists an $r$-successor $y$ of

$x$ in $\mathcal{I}$ with $y \in \mathsf{val}_r(C_i)^{\mathcal{I}} \subseteq D''^{\mathcal{I}}$. Thus, $x \in (\exists r.D'')^{\mathcal{I}}$. $\mathsf{ex}_r(C_i) \neq \emptyset$. The induced existential restrictions for $C_i$ are $\mathsf{ind\text{-}ex}_r(C_i) = \left\{ \bigsqcup_{1 \leq l \leq k_i} (C_{il} \sqcap \mathsf{val}_r(C_i)) \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \right\}$. Since $C_i$ is consistent, we can consider a canonical tree model $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $C_i$. Let $x \in C_i^{\mathcal{I}}$. Hence, $x$ has $n = min\{\mathsf{max}_r(C_i), |\mathsf{ex}_r(C_i)|\}$ $r$-successors $y_1, \ldots, y_n$. Each $y_j \in E_j^{\mathcal{I}}$ s.t. $E_j = \bigsqcap_{B \in \mathsf{ex}_r(C_i)} B$, where for all $E_j$ holds (1) $E_j \not\equiv \bot$, (2) $E_j \neq \emptyset$ and (3) $\{E_j \mid 1 \leq j \leq n\} = \mathsf{ex}_r(C_i)$. In other words, $\{E_j \mid 1 \leq j \leq n\}$ is a set of merged concept descriptions. Hence, there exists $\alpha' \in \Gamma_r(C_i)$ s.t. $\mathsf{ex}_r(C_i)^{\alpha'} = \{E_j \mid 1 \leq j \leq n\}$.

By Condition 4, there exists $C' \in \mathsf{ind\text{-}ex}_r(C_i)$ s.t. $C' \sqcap \mathsf{val}_r(C_i) \sqsubseteq D''$ and since $\mathsf{ind\text{-}ex}_r(C_i) = \left\{ \bigsqcup_{1 \leq l \leq k_i} (C_{il} \sqcap \mathsf{val}_r(C_i)) \mid C_{ij} \in \mathsf{ex}_r(C_i)^{\alpha_{ij}}, 1 \leq j \leq k_i \right\}$, there exists for each $\alpha_i \in \{\alpha_1, \ldots, \alpha_k\}$ a $C'' \in \mathsf{ex}_r(C_i)^{\alpha_i}$ s.t. $C'' \sqsubseteq C'$. For $\alpha'$ let $E_k$ be the concept from $\mathsf{ex}_r(C_i)^{\alpha'}$ with $E_k \equiv C''$, thus $y_k \in E_k^{\mathcal{I}} \equiv C''^{\mathcal{I}} \subseteq C'^{\mathcal{I}} \subseteq D''^{\mathcal{I}}$. We obtain $x \in \exists r.D''^{\mathcal{I}}$.

- Let $D' = \forall r.D''$: If $\mathsf{max}_r(C_i) = 0$, then $\mathsf{ind\text{-}val}_r(C_i) = \bot$ and there exists no $r$-successor of $x$ in $\mathcal{I}$, and $x \in (\forall r.D'')^{\mathcal{I}}$.

  Assume $\kappa_r(C_i) \leq \mathsf{max}_r(C_i)$. Thus the induced value restriction $\mathsf{ind\text{-}val}_r(C_i) = \mathsf{val}_r(C_i)$. Let $y$ be an arbitrary $r$-successor of $x \in \mathcal{I}$, then $x \in C^{\mathcal{I}}$ implies $y \in \mathsf{val}_r(C)^{\mathcal{I}}$ and because of Condition 5, $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(D')$, we obtain $y \in \mathsf{val}_r(D')^{\mathcal{I}}$. Thus $x \in (\forall r.D'')^{\mathcal{I}}$.

  Finally, assume $0 \leq \kappa_r(C_i) = \mathsf{max}_r(C_i)$. Thus the induced value restriction $\mathsf{ind\text{-}val}_r(C_i) = \bigsqcup_{C' \in \mathsf{ex}_r(C_i)^*} (\mathsf{val}_r(C_i) \sqcap C')$. Since $0 \leq \kappa_r(C_i) = \mathsf{max}_r(C_i)$ we know there exist exactly $\kappa_r(C_i)$ distinct $r$-successors of $x$ in $\mathcal{I}$. Since $C_i$ is consistent, we can consider a canonical tree model $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $C_i$. Let $x \in C_i^{\mathcal{I}}$. Hence, $x$ has exactly $n = \mathsf{max}_r(C_i) = \kappa_r(C_i)$ distinct $r$-successors $y_1, \ldots, y_{\kappa_r(C_i)}$. Each $y_j \in E_j^{\mathcal{I}}$ s.t. $E_j = \bigsqcap_{B \in \mathsf{ex}_r(C_i)} B$, where for all $E_j$ holds (1) $E_j \not\equiv \bot$, (2) $E_j \neq \emptyset$ and (3) $\{E_j \mid 1 \leq j \leq n\} = \mathsf{ex}_r(C_i)$. In other words, $\{E_j \mid 1 \leq j \leq n\}$ is a set of merged concept descriptions. Hence, there exists $\alpha' \in \Gamma_r(C_i)$ s.t. $\mathsf{ex}_r(C_i)^{\alpha'} = \{E_j \mid 1 \leq j \leq \kappa_r(C_i)\}$. Since $E_j \in \mathsf{ex}_r(C_i)^{\alpha'}$, we know $E_j \in \bigcup_{\alpha \in \Gamma_r(C_i)} \mathsf{ex}_r(C_i)^{\alpha}$ and thus $E_j \in \mathsf{ex}_r(C_i)^*$.

  Since $\mathcal{I}$ is a model, $y_j \in \mathsf{val}_r(C_i)^{\mathcal{I}}$. Thus $y_j \in (\mathsf{val}_r(C_i) \sqcap E_j)^{\mathcal{I}}$. Consequently, $y_i \in (\bigsqcup_{E_j \in \mathsf{ex}_r(C_i)^*} \mathsf{val}_r(C_i) \sqcap E_j)^{\mathcal{I}} = \mathsf{ind\text{-}val}_r(C_i)^{\mathcal{I}}$ for all $1 \leq j \leq \kappa_r(C_i)$. Thus $x \in (\forall r.\mathsf{ind\text{-}val}_r(C_i))^{\mathcal{I}}$. By Condition 5 we have $x \in (\forall r.\mathsf{ind\text{-}val}_r(D'))^{\mathcal{I}}$.

We have shown that $C_i \sqsubseteq D'$ for arbitrary $C_i$ and each top-level conjunct $D'$ of $D$ and, consequently, $\bigsqcup_{1 \leq i \leq n} C_i \sqsubseteq D$ holds. ❑

By the conditions from Theorem 45 we have shown that the information captured in $prim, \mathsf{max}_r, \mathsf{min}_r, \mathsf{ind\text{-}ex}_r$ and $\mathsf{ind\text{-}val}_r$ of a concept description in $\mathcal{ALCN}$-normal form is all the information implied by this concept. Based on these formal constructs for implied information, we can devise an algorithm to compute $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions.

Input: $\mathcal{ALCN}$-concept description $C$. Output: $\mathcal{ALEN}$-approximation of $C$.

1. If $C \equiv \bot$ then $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) := \bot$ or
   if $C \equiv \top$ then $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) := \top$.

2. Otherwise, transform $C$ into $\mathcal{ALCN}$-normal form $C_1 \sqcup \cdots \sqcup C_n$ and return

$$\begin{aligned}
&\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) := \\
&\quad \sqcap_{A \in \bigcap_i prim(C_i)} A \\
&\quad \sqcap\ (\geq \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}\ r) \\
&\quad \sqcap\ (\leq \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}\ r) \\
&\quad \sqcap \bigsqcap_{\substack{(C'_1,\ldots,C'_n) \in \\ \mathsf{ind\text{-}ex}_r(C_1) \times \cdots \times \mathsf{ind\text{-}ex}_r(C_n)}} \exists r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_i \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}) \\
&\quad \sqcap\ \forall r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{ind\text{-}val}_r(C_i)) \mid 1 \leq i \leq n\})
\end{aligned}$$

Figure 4.2: The algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$.

### 4.2.3 Computing $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions

The algorithm for computing $\mathcal{ALEN}$-approximations from $\mathcal{ALCN}$-concept descriptions is an extension of the algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$. For this kind of extension we have to use the $\mathcal{ALCN}$-normal form, take into account the implied information computed according the notions defined in Section 4.2.1 and then also extract the number restrictions common to all disjuncts in the concept description that is to be approximated. The resulting algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ is depicted in Figure 4.2.

The following theorem states that $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ finds in fact the approximation according to Definition 30, where $\mathcal{L}_1$ is $\mathcal{ALCN}$ and $\mathcal{L}_2$ is $\mathcal{ALEN}$.

**Theorem 46.** *Let $C$ be an $\mathcal{ALCN}$-concept description in $\mathcal{ALCN}$-normal form. Then $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ is the $\mathcal{ALEN}$-approximation of $C$, i.e.,*

1. *$C \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$; and*
2. *$\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqsubseteq E$ for every $\mathcal{ALEN}$-concept description $E$ with $C \sqsubseteq E$.*

*In particular, $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions always exist and can be computed effectively.*

Next we have to show the correctness of the algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$. More precisely, we have to show that the concept description returned by $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$, if applied to a concept $C$, subsumes $C$ and is the most specific $\mathcal{ALEN}$-concept description with this property. One can prove $C \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ by structural induction on $C$ using Theorem 45, where $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ takes the place of $D$. In order to prove minimality of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ (w.r.t. subsumption), one assumes another $\mathcal{ALEN}$-concept $E$ subsuming $C$. Again, using Theorem 45, one can show that $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqsubseteq E$.

**Proof. (Theorem 46)** 1. We first show that $C \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ by proof by induction over the structure of $C$. If $C \in \{\bot, \top\}$ then $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) = C$, trivially implying subsumption. Otherwise, the induction hypothesis guarantees that the claim holds for the subterms of $C$ occurring in existential and value restrictions. It is sufficient to prove that the conditions presented in Theorem 45 hold.

- By definition of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ it holds that $prim(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)) = \bigcap_{i=1}^{n} prim(C_i) \subseteq prim(C)$.
- By definition of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ it holds that $\mathsf{max}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)) = \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}$, where obviously $\max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\} \geq \mathsf{max}_r(C_i)$ for all $1 \leq i \leq n$.
- By definition of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ it holds that $\mathsf{min}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)) = \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}$, where obviously $\min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\} \leq \mathsf{min}_r(C_i)$ for all $1 \leq i \leq n$.
- For every existential restriction $\exists r.\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_i' \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big)$ in $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$, $C_i' \in \mathsf{ind\text{-}ex}_r(C_i)$ for every $i$. By induction hypothesis, $C_i' \sqcap \mathsf{val}_r(C_i) \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_i' \sqcap \mathsf{val}_r(C_i))$. Thus, by definition of the lcs, $C_i' \sqcap \mathsf{val}_r(C_i) \sqsubseteq \mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_i' \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big)$.
- Show: $\mathsf{val}_r(C) \sqsubseteq \mathsf{val}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C))$. By induction hypothesis we already know that $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{val}_r(C_i))$ for every $i$. Consequently, for the lcs we find $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big)$.

2. Without loss of generality, let $E$ be in $\mathcal{ALEN}$-normal form. We give a proof by induction over the structure of $C$. If $C \in \{\bot, \top\}$, then $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) = C$ which is the least concept subsuming $C$. Otherwise, we may assume that the claim holds for the subterms of $C$ occurring in existential and value restrictions. If $E = \top$, then trivially $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqsubseteq E$. Otherwise, the subsumption $C \sqsubseteq E$ induces the following facts:

- $prim(E) \subseteq prim(C_i)$ for every $i$. Since $prim(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C))$ is the intersection of every $prim(C_i)$, this implies $prim(E) \subseteq prim(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C))$.
- $\mathsf{max}_r(C_i) \leq \mathsf{max}_r(E)$ for every $i$. Since $\mathsf{max}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)) = \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}$ this implies $(\geq \mathsf{max}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C))\ r) \sqsubseteq (\geq \mathsf{max}_r(E)\ r)$.
- $\mathsf{min}_r(E) \leq \mathsf{min}_r(C_i)$ for every $i$. Since $\mathsf{min}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)) = \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}$ this implies $(\leq \mathsf{min}_r(\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C))\ r) \sqsubseteq (\leq \mathsf{min}_r(E)\ r)$.
- For all $E' \in \mathsf{ex}_r(E)$ and for all $i$ there is one $C' \in \mathsf{ex}_r(C_i)$ with $C' \sqcap \mathsf{val}_r(C_i) \sqsubseteq E'$. The induction hypothesis now guarantees that $C' \sqcap \mathsf{val}_r(C_i) \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C' \sqcap \mathsf{val}_r(C_i)) \sqsubseteq E'$ for every $i$. Hence, for the lcs it holds that

$$\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C' \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big) \sqsubseteq E'.$$

- For all $i$ we have $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{val}_r(E)$. By induction hypothesis, we have $\mathsf{val}_r(C_i) \sqsubseteq \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{val}_r(C_i)) \sqsubseteq \mathsf{val}_r(E)$. Thus, we find $\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}\big) \sqsubseteq \mathsf{val}_r(E)$.

These facts imply $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqsubseteq E$. ❑

By omitting some of the sub-concepts computed by $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$, we obtain approximations in sublanguages of $\mathcal{ALEN}$. For example, if we discard the number restrictions computed by $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$, we obtain an $\mathcal{ALE}$-approximation of the given $\mathcal{ALCN}$-concept description.

Concerning the computational complexity of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ it should be noted that the complexity of computing the lcs of $\mathcal{ALEN}$-concept descriptions yields a lower bound since the $\mathcal{ALEN}$-approximation of $C_1 \sqcup C_2$ for two $\mathcal{ALEN}$-concept descriptions $C_1$ and $C_2$ is exactly the lcs of $C_1$ and $C_2$ in $\mathcal{ALEN}$. The computation of the lcs of $\mathcal{ALEN}$-concept descriptions takes double-exponential time in the worst-case [KM01b]. To compute the resulting existential restrictions, $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ furthermore must consider a possibly exponential number of tuples $(C'_1, \ldots, C'_n)$ and a possibly exponential number of existential mergings. Therefore, $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ is at least a double-exponential time algorithm. However, it is not known whether the bound for the lcs is tight, and whether tight complexity bounds for the lcs would carry over to the approximation problem.

## 4.3 Difference operator for $\mathcal{ALC}$-$\mathcal{ALE}$

We have seen how to compute the $\mathcal{ALE}$-approximation of a given $\mathcal{ALC}$-concept description. For such a pair $C, D$ of approximated and approximating concepts, a natural question regards the loss of information, i.e., what aspects of $C$ are not captured by $D$. An answer to such questions requires a notion of the 'difference' between concept descriptions. For instance, a comparison between the example concept $C_{\text{ex}}$ from Example 33 on page 64 and its approximation $\exists r.(A \sqcap \neg B) \sqcap \exists r.(B \sqcap \neg A)$ should reveal that the value restriction $\forall r.(\neg A \sqcup \neg B)$ is not captured by its approximation.

A first approach for a difference-operator has been proposed by Teege in [Tee94]. Here, the difference $C - D$ of two given $\mathcal{L}$-concept descriptions with $C \sqsubseteq D$ has been defined as

$$\max\{E \in \mathcal{L} \mid E \sqcap D \equiv C\}$$

where the maximum is defined w.r.t. subsumption. Since $\mathcal{ALC}$ provides full negation, the most general concept description $E$ with $E \sqcap D \equiv C$ is always $C \sqcup \neg D$. Consequently, the difference operator proposed by Teege would return

$$(\exists r.A \sqcap \exists r.B \sqcap \forall r.(\neg A \sqcup \neg B)) \sqcup \neg(\exists r.(A \sqcap \neg B) \sqcap \exists r.(B \sqcap \neg A))$$

as the difference between $C_{\text{ex}}$ and its approximation, which does not help a user to ascertain the information lost by the approximation. The example illustrates that it might be promising to look for a syntactic minimum instead of a semantic maximum in order to find a compact representation of the difference of two concepts.

In [Küs01], a so-called *sub-description ordering* has been proposed to deal with syntactical redundancies. The first step to extend to this approach to our case is to introduce such an ordering on $\mathcal{ALC}$-concept descriptions. The idea is to obtain a sub-description of some $\mathcal{ALC}$-concept description $C$ by means of two kinds of modifications. Firstly, by making inconsistencies explicit; and secondly, by removing disjuncts and conjuncts, and by replacing some existential or value restrictions by their respective sub-descriptions. Formally, this leads to the following definition.

**Definition 47 (Sub-description ordering ($\preceq_d$)).** Let $C, D$ be $\mathcal{ALC}$-concept descriptions in $\mathcal{ALC}$-normal form. Let $C = C_1 \sqcup \cdots \sqcup C_n$. Then, $D \preceq_d C$ iff $D \in \{\bot, \top\}$ or $D$ is obtained from $C$ by performing some of the following steps.

1. Remove some disjuncts $C_i$ for $1 \leq i \leq n$,
2. for every remaining $C_i$:
   (a) remove some conjuncts $A \in \mathit{prim}(C_i)$,
   (b) remove some conjuncts $\exists r.C_i'$ with $C_i' \in \mathsf{ex}_r(C_i)$,
   (c) remove the conjunct $\forall r.\mathsf{val}_r(C_i)$,
   (d) for every remaining $C_i' \in \mathsf{ex}_r(C_i) \cup \{\mathsf{val}_r(C_i)\}$: replace $C_i'$ by $C_i''$ with $C_i'' \preceq_d C_i'$.

◇

As an example, consider the concept descriptions $C := \exists r.A \sqcap \forall r.\neg B$ and $D := (\exists r.(A \sqcup B) \sqcap \forall r.\neg B) \sqcup (\exists r.\neg A \sqcap \forall r.A)$. By removing the last disjunct from $D$ and removing the last disjunct in the remaining existential restriction we find that $C \preceq_d D$. Note that $C \equiv D$. Based on the sub-description ordering, we can provide the definition of the syntactic difference operator.

**Definition 48 (Difference operator).** Let $C$ be an $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form and $D$ be an $\mathcal{ALE}$-concept description in $\mathcal{ALE}$-normal form. Then, the $\mathcal{ALC}$-concept description $E$ is called the *difference of $C$ and $D$*, ($C - D$ for short), iff

1. $E \sqcap D \equiv C \sqcap D$, and
2. for every $\mathcal{ALE}$-concept description $E'$ with $E' \sqcap D \equiv C \sqcap D$ it holds that either $E \preceq_d E'$ or $E$ and $E'$ are incomparable with respect to $\preceq_d$.

◇

Intuitively, the idea is to remove all sub-descriptions from $C$ which are either redundant in $C$ or already present in $D$. It should be noted that in case of $C \sqsubseteq D$, and thus, $C \sqcap D \equiv C$, the only difference to Teege's difference operator is that the minimum w.r.t. $\preceq_d$ is used instead of the maximum w.r.t. $\sqsubseteq$. In general, the difference $C - D$ is not maximal with respect to subsumption, as a simple example illustrates. For $C = A \sqcup B$ and $D = B$, we obtain $C - D = A$, although $C \sqcap (A \sqcup B) \equiv C \sqcap D$, i.e., $A \sqcup B$ is a more general solution w.r.t. subsumption.

Input: $\mathcal{ALC}$-concept description $C = C_1 \sqcup \cdots \sqcup C_n$ in $\mathcal{ALC}$-normal form,
$\mathcal{ALE}$-concept description $D$

Output: $\mathsf{c\text{-}diff}(C, D)$

1. If $C \sqcap D \equiv \bot$, then $\mathsf{c\text{-}diff}(C, D) := \bot$;
2. If $n > 1$, then let $\mathsf{c\text{-}diff}(C, D) := \bigsqcup_{i=1}^{n} \mathsf{c\text{-}diff}(C_i, D)$ and iteratively remove $\mathsf{c\text{-}diff}(C_j, D)$ from the disjunction in case $\mathsf{c\text{-}diff}(C_j, D) \sqsubseteq \bigsqcup_{i \neq j} \mathsf{c\text{-}diff}(C_i, D)$;
3. If $n = 1$, then $\mathsf{c\text{-}diff}(C, D) :=$

$$\prod_{A \in prim(C) \setminus prim(D)} A \;\sqcap\; \forall r.\mathsf{c\text{-}diff}(\mathsf{val}_r(C), \mathsf{val}_r(D)) \;\sqcap\; \prod_{E \in \mathcal{E}'_r} \exists r.E$$

where

- value restriction is omitted in case $\mathsf{c\text{-}diff}(\mathsf{val}_r(C), \mathsf{val}_r(D)) \equiv \top$
- $\mathcal{E}'_r$ is computed as follows:
  Let $\mathcal{E}_r = \{C'_1, \ldots, C'_n\} := \mathsf{ex}_r(C)$.
  For $i = 1$ to $n$ do begin
  If (i) there exists $C' \in \mathcal{E}_r \setminus \{C'_i\}$ with $\mathsf{val}_r(D) \sqcap \mathsf{val}_r(C) \sqcap C' \sqsubseteq C'_i$, or
  (ii) there exists $D' \in \mathsf{ex}_r(D)$ with $\mathsf{val}_r(D) \sqcap \mathsf{val}_r(C) \sqcap D' \sqsubseteq C'_i$
  then $\mathcal{E}_r := \mathcal{E}_r \setminus \{C'_i\}$
  end

  $\mathcal{E}'_r := \{E^* \mid E \in \mathcal{E}_r\}$, where

$$E^* := \begin{cases} \mathsf{c\text{-}diff}(E, \mathsf{val}_r(C) \sqcap \mathsf{val}_r(D)), & \text{if } \mathsf{val}_r(C) \text{ is an} \\ & \mathcal{ALE}\text{-concept description,} \\ \mathsf{c\text{-}diff}(E, \mathsf{val}_r(D)) & \text{otherwise.} \end{cases}$$

Figure 4.3: The algorithm $\mathsf{c\text{-}diff}(C, D)$.

Finally, it should be noted that a priori the difference between $C$ and $D$ is not uniquely determined. By abuse of language and notation, we will still refer to *the* difference $C - D$. Coming back to the example from the beginning of this subsection, the difference (according to Definition 48) between $C_{\text{ex}}$ and its approximation is in fact the desired value restriction $\forall r.(\neg A \sqcup \neg B)$.

Having defined our difference operator, we need to devise an algorithm to actually compute the difference $C - D$. In [Küs01], an algorithm has been proposed to compute the difference $C - D$ of $\mathcal{ALE}$-concept descriptions $C$ and $D$. Extending this algorithm to the case of $\mathcal{ALC}$-concept descriptions $C$ yields our definition of the algorithm $\mathsf{c\text{-}diff}$ as depicted in Figure 4.3.

If $C$ is a disjunction of sub-concepts $C_i$ then the difference between $C$ and $D$ is computed by first computing the differences between the disjuncts and $D$ and then

eliminating the semantically redundant resulting disjuncts. If $n = 1$, $C$ is a conjunction of $\mathcal{ALC}$-concept descriptions (with possibly just one conjunct). In this case, redundant concept names and existential restrictions on the top-level conjunction of $C$ are removed. Furthermore, redundancies in existential restrictions and value restrictions are removed recursively. The set $\mathcal{E}'_r$ can be computed by iteratively removing existential restrictions of $C$ that do not satisfy Conditions 3(i) or 3(ii). Given an oracle for subsumption, this can be carried out in polynomial time.

The following proposition proves that $\mathsf{c\text{-}diff}(C, D)$ respects the first condition of the difference operator (Definition 48), i.e., it does not remove too much from the original concept description $C$.

**Proposition 49.** *Let $C$ be an $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form and $D$ an $\mathcal{ALE}$-concept description in $\mathcal{ALE}$-normal form. Then, $\mathsf{c\text{-}diff}(C, D) \sqcap D \equiv C \sqcap D$.*

**Proof.** We give a proof by induction over the structure of $C$.

1. $C \in \mathit{prim}(C)$
   As $\mathit{prim}(\mathsf{c\text{-}diff}(C, D)) = \mathit{prim}(C) \setminus \mathit{prim}(D)$, it follows that $\mathsf{c\text{-}diff}(C, D) \sqcap D$ is equivalent to $D \sqcap \bigsqcap_{A \in \mathit{prim}(\mathsf{c\text{-}diff}(C,D))} A$. We can safely add to this another conjunct more general than $D$ yielding
   $$D \sqcap \bigsqcap_{A \in \mathit{prim}(\mathsf{c\text{-}diff}(C,D))} A \sqcap \bigsqcap_{A \in \mathit{prim}(C) \cap \mathit{prim}(D)} A.$$
   The expression thus obtained is equivalent to $C \sqcap D$.

2. $C = C_1 \sqcup C_2$
   Without loss of generality, assume exactly two disjuncts on the top-level of $C$. By definition, even after removing redundant disjuncts, $\mathsf{c\text{-}diff}((C_1 \sqcup C_2), D)$ is equivalent to $\mathsf{c\text{-}diff}(C_1, D) \sqcup \mathsf{c\text{-}diff}(C_2, D)$. Hence, the conjunction $\mathsf{c\text{-}diff}((C_1 \sqcup C_2), D)$ with $D$ can be simplified to $\mathsf{c\text{-}diff}(C_1, D) \sqcap D \sqcup \mathsf{c\text{-}diff}(C_2, D) \sqcap D$. According to the induction hypothesis, this yields $(C_1 \sqcap D) \sqcup (C_2 \sqcap D)$, which simplifies to $(C_1 \sqcup C_2) \sqcap D$.

3. No disjunction on the top-level of $C$
   Show $\mathsf{c\text{-}diff}(C, D) \sqcap D \equiv C \sqcap D$. According to the characterization of subsumption (Theorem 36), three conditions must hold for equivalence:

   - The set $\mathit{prim}(\mathsf{c\text{-}diff}(C, D) \sqcap D)$ equals $\mathit{prim}(\mathsf{c\text{-}diff}(C, D)) \cup \mathit{prim}(D)$ which by definition is $(\mathit{prim}(C) \setminus \mathit{prim}(D)) \cup \mathit{prim}(D)$. This is equal to $\mathit{prim}(C) \cup \mathit{prim}(D)$, the set of primitive concepts in $C \sqcap D$.
   - By induction hypothesis, $\mathsf{c\text{-}diff}(\mathsf{val}_r(C), \mathsf{val}_r(D)) \sqcap \mathsf{val}_r(D)$ is equivalent to $\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D)$. By definition $\mathsf{val}_r(C \sqcap D)$ is equivalent to $\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D)$ which concludes this case.
   - Show ($\sqsubseteq$). Let $F' \in \mathsf{ex}_r(C \sqcap D)$. We have to find an $E' \in \mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D)$ with $E' \sqcap \mathsf{val}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D) \sqsubseteq F'$. From the previous case we know that $\mathsf{val}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D)$ is equivalent to $\mathsf{val}_r(C \sqcap D)$. Since $\mathsf{ex}_r(C \sqcap D)$ is equal to the union $\mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D)) \cup \mathsf{ex}_r(D)$ we may distinguish two cases.

If $F' \in \mathsf{ex}_r(D)$ then we can select $E' := F'$, because it also occurs in the set $\mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D)$ which is the conjunction of the concept descriptions in $\mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D)) \cup \mathsf{ex}_r(D)$. We thus obviously find $E' \sqcap \mathsf{val}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D) \sqsubseteq F'$.

If $F' \in \mathsf{ex}_r(C) \backslash \mathsf{ex}_r(D)$, then Conditions 3(i) and 3(ii) in the definition of the algorithm $\mathsf{c\text{-}diff}(C, D)$ guarantee that there exists an existential restriction $\tilde{E}' \in \mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D))$ with the following properties. If $\mathsf{val}_r(C)$ is an $\mathcal{ALE}$-concept description then $\tilde{E}'$ is of the form $\mathsf{c\text{-}diff}(E', (\mathsf{val}_r(D) \sqcap \mathsf{val}_r(C)))$ with $E' \sqcap \mathsf{val}_r(D) \sqcap \mathsf{val}_r(C) \sqsubseteq F'$. According to the induction hypothesis, $\mathsf{c\text{-}diff}(E', (\mathsf{val}_r(D) \sqcap \mathsf{val}_r(C))) \sqcap \mathsf{val}_r(D) \sqcap \mathsf{val}_r(C)$ is equivalent to $E' \sqcap \mathsf{val}_r(D) \sqcap \mathsf{val}_r(C)$. Consequently, we find that $\tilde{E}' \sqcap \mathsf{val}_r(C) \sqcap \mathsf{val}_r(D) \sqsubseteq F'$. It is easy to see that $\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D)$ is equivalent to $\mathsf{val}_r(C \sqcap D)$ which again is equivalent to $\mathsf{val}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D)$ as we know from above. Hence, we have found an $\tilde{E}'$ with $\tilde{E}' \sqcap \mathsf{val}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D) \sqsubseteq F'$. If $D$ is no $\mathcal{ALE}$-concept description then $\tilde{E}'$ is of the form $E' \sqcap \mathsf{val}_r(D)$. This case is analogous to the previous one.

Show ($\sqsupseteq$). In analogy to the case ($\sqsubseteq$), consider some $E' \in \mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D) \sqcap D)$. We have to find an $F' \in \mathsf{ex}_r(C \sqcap D)$ such that $F' \sqcap \mathsf{val}_r(C \sqcap D) \sqsubseteq E'$. Again, we have two cases to distinguish.

If $E' \in \mathsf{ex}_r(D)$, then we can again select $F' := E'$ which also occurs in $\mathsf{ex}_r(C \sqcap D)$.

If $E' \in \mathsf{ex}_r(\mathsf{c\text{-}diff}(C, D)) \setminus \mathsf{ex}_r(D)$, then Condition 3(ii) guarantees that an $F' \in \mathsf{ex}_r(D) \subseteq \mathsf{ex}_r(C \sqcap D)$ exists such that $F' \sqcap \mathsf{val}_r(C) \sqcap \mathsf{val}_r(D) \sqsubseteq E'$. As shown above, $\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D)$ is equivalent to $\mathsf{val}_r(C \sqcap D)$ which concludes the argument.

❑

It remains to examine the computational complexity of the algorithm c-diff. In the following corollary it is shown that c-diff is a polynomial time algorithm.

**Corollary 50.** *Given an oracle for subsumption, the algorithm* c-diff *is a polynomial time algorithm, i.e., for a given $\mathcal{ALC}$-concept description $C$ in $\mathcal{ALC}$-normal form and an $\mathcal{ALE}$-concept description $D$, the computation of* $\mathsf{c\text{-}diff}(C, D)$ *takes at most polynomial time in the size of $C$ and $D$.*

**Proof.** It is not difficult to see that the size of the output $\mathsf{c\text{-}diff}(C, D)$ never exceeds the size of $C$: if $n > 1$ then the difference is simply distributed to the disjuncts, and if $n = 1$ then, (1) some primitive concepts are removed, thus reducing the size of the resulting concept description, (2) the value restriction is handled recursively and (3) some existential restrictions are removed while the remaining ones are also handled recursively. Consequently, during the recursive computation of $\mathsf{c\text{-}diff}(C, D)$ the algorithm is never applied to an argument exceeding the size of the input. Neither does the algorithm introduce new existential or value restrictions during the computation of $\mathsf{c\text{-}diff}(C, D)$.

Thus, it is sufficient for our claim to show that (1) the computation of the subset $\mathcal{E}'_r$ takes only polynomial time in the size of the input and (2) there are at most poly-

nomially many (in the size of $C$ and $D$) calls to c-diff during the recursive computation of c-diff$(C, D)$.

1. As the condition in Step 3 states, an appropriate subset $\mathcal{E}'_r$ can be found by iteratively removing elements from the original set $\mathsf{ex}_r(C)$ and verifying Conditions 3(i) and 3(ii) in every iteration. Thus, the number of subsets to inspect is bounded by the size of $C$. For every subset, a polynomial number of subsumption test must be made. Given an oracle for subsumption, this task requires only polynomial time.

2. Recursive calls to the algorithm c-diff are necessary for the computation of c-diff$(\mathsf{val}_r(C), \mathsf{val}_r(D))$ as well as for the computation of every $E^*_j$. Nevertheless, there is only one value restriction $\mathsf{val}_r(C)$ in $C$ the size of which is bounded by the size of $C$. As no new value restrictions are introduced, we have at most polynomially many expressions of the form c-diff$(\mathsf{val}_r(C), \mathsf{val}_r(D))$ to evaluate during the execution of c-diff$(C, D)$.

   As c-diff does not introduce new existential restrictions and as the size of its output never exceeds the size of its input, it is easy to see that the number of existential restrictions $E'_j$ and their size is bounded by the input. Consequently, the number of calls to c-diff is bounded by the syntax tree of the input concept $C$ which again is bounded by the size of $C$, since $C$ was assumed in $\mathcal{ALC}$-normal form.

❑

It should be recalled though that transforming an arbitrary $\mathcal{ALC}$-concept description into $\mathcal{ALC}$-normal form can produce an exponentially larger concept description. To summarize the existing results, the following properties can be shown for every computation of the algorithm c-diff$(C, D)$.

**Theorem 51.** *Let $C$ be an $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form and $D$ be an $\mathcal{ALE}$-concept description. Then,* c-diff$(C, D)$ *satisfies all of the following properties:*

1. c-diff$(C, D) \sqcap D \equiv C \sqcap D$,

2. *If $C$ is an $\mathcal{ALE}$-concept description, then $C-D$ is uniquely determined modulo associativity and commutativity of concept conjunction, and $C-D$ and* c-diff$(C, D)$ *coincide modulo associativity and commutativity, and*

3. *Given an oracle for subsumption, the computation of* c-diff$(C, D)$ *takes polynomial time in the size of $C$ and $D$.*

The first property where only $\mathcal{ALE}$ is considered was already shown in [Küs01]. The others have been shown in the above proposition and corollary.

The algorithm c-diff is only a heuristic for the difference operator, since it does not fulfill the second property of Definition 48. The main reason is the recursive call of c-diff to obtain $E^* :=$ c-diff$(E, \mathsf{val}_r(C) \sqcap \mathsf{val}_r(D))$. In general, $C$ and thus also $\mathsf{val}_r(C)$ is an $\mathcal{ALC}$-concept description to which the difference operator $\mathcal{ALC}$-$\mathcal{ALE}$ is not applicable. This can be remedied by an *improved heuristic for* c-diff to a certain

extent by using approximation for the recursive call on $\mathcal{ALC}$-concept descriptions in $\mathsf{val}_r(C)$. The last line of the c-diff algorithm in Figure 4.3 would thus be changed to: $\mathsf{c\text{-}diff}(E, \mathsf{c\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D))$ for the improved heuristic. Due to the use of approximation, c-diff would no longer be a polynomial time algorithm (with an oracle for subsumption).

Besides for the application for assessing the information loss between a concept description and its approximation, the difference operator can also be utilized to obtain a compact representation of a concept description. If applying $\mathsf{c\text{-}diff}(C, \top)$ to an $\mathcal{ALC}$-concept description $C$, the computed concept description is equivalent to $C$ with redundancies removed.

In this chapter we have proposed an approach to obtain a meaningful lcs by using concept approximation as a pre-processing step. This pre-processing step translates the input concept descriptions from a DL with disjunction to a DL that does not offer disjunction and where the lcs exists. Methods to realize this approach have been devised and proven correct for two such pairs of DLs. In particular, it has been shown that the $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concept descriptions always exists and can be computed effectively. An algorithm for computing approximation has been devised that runs in 2-ExpTime. This method has been extended by number restrictions. For $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions it has been shown that the approximation always exists and a computation algorithm has been devised that is worst case double-exponential. Equipped with these methods, a common subsumer can now be obtained that captures the commonalities of concept descriptions that use disjunction in an explicit way.

The methods proposed here can be applied to obtain meaningful common subsumers in even less expressive DLs. For instance, the commonalities of $\mathcal{ALU}$-concept descriptions[18] could be computed by:

1. compute the $\mathcal{ALE}$-approximation of the concept descriptions
2. compute their lcs and,
3. replace qualified existential restrictions by the unqualified one.

Similar to the case of computing least common subsumers the method proposed in this section to compute concept approximations can be employed to obtain 'approximative' results of other reasoning methods for which computation methods so far only exists for the target DLs of approximation. Furthermore, with the computation methods for approximation simplified views of DL knowledge bases that are written in an expressive DL can be computed and then displayed to the user for easier comprehension.

Computing the approximation of a concept description results in information loss in general. The loss of information can be assessed by the syntactic difference operator. We have investigated a method to compute the syntactic difference between

[18] $\mathcal{ALU}$-concept descriptions can be build from conjunction, disjunction, unqualified existential restriction ($\exists r.\top$) and value restrictions.

$\mathcal{ALC}$-concept descriptions and $\mathcal{ALE}$-concept descriptions. However, this method does not find an optimal solution, but computes only a heuristic. Thus it reveals only some of the information lost; information on value restriction that was lost during approximation might remain undiscovered by applying c-diff. Nevertheless, by this method we can compute the difference of a concept description and its approximation to see what information was lost during approximation. Regarding the lcs, one can employ the difference operator to find out which parts of the input concepts are not captured in the commonalities expressed by the lcs (of the approximated concept).

This sums up our theoretical results on the approximation-based approach. Next, we consider the computation of common subsumers w.r.t. a background ontology to obtain non-trivial common subsumers in the presence of disjunction.

# Chapter 5

# Common subsumers w.r.t. a background terminology

In this chapter we discuss the second approach to generalize concept descriptions in a meaningful way for expressive DLs that offer disjunction. This approach is employed in applications where a TBox in an expressive DL is customized, i.e., extended by another TBox written in a less expressive DL. For such an application, the lcs operator is applied to concept descriptions using the concept operators from the 'smaller' DL referring to concept names defined in the TBox written in the expressive DL.

The underlying framework and first results on the lcs w.r.t. background terminologies have been published by Baader, Sertkaya and Turhan in [BST04b] and these results were later extended in [BST04a; BST07]. Next we introduce the framework for customizing a background ontology.

## 5.1 Framework for customizing background ontologies

In this framework we assume that there is a fixed *background terminology* defined in an expressive DL, e.g., a large ontology written by experts, which the user has bought from some ontology provider. The user then wants to extend this terminology in order to adapt it to the needs of a particular application domain. However, since the user is not a DL expert, she employs a less expressive 'user DL' and needs support through the bottom-up approach when building this user-specific extension of the background terminology. There are several reasons to employ a restricted DL in this setting: such a restricted DL may be easier to comprehend and to use for a non-expert; it may allow for a more intuitive graphical or frame-like user interface; and to use the bottom-up approach, the lcs must exist and make sense, and it must be possible to compute it with reasonable effort. Thus we assume that the user DL does not offer concept disjunction. Recall from Section 1.2.2 that the lcs obtained for the selected concepts is then (edited by the user and) added to the user terminology, see Figure 1.2 on page 13.

To make this more precise, consider a background terminology (a TBox) $\mathcal{T}$ defined in an expressive DL $\mathcal{L}_2$. When defining new concepts, the user employs only

a sublanguage $\mathcal{L}_1$ of $\mathcal{L}_2$ for which computing the lcs makes sense and does not simply build a disjunction of the input concept descriptions. However, in addition to primitive concepts and roles, the concept descriptions written in the DL $\mathcal{L}_1$ may also contain names of concepts defined in $\mathcal{T}$. Let us call such concept descriptions $\mathcal{L}_1(\mathcal{T})$-concept descriptions. Depending on the DLs $\mathcal{L}_1$ and $\mathcal{L}_2$, least common subsumers of $\mathcal{L}_1(\mathcal{T})$-concept descriptions w.r.t. an $\mathcal{L}_2$-TBox $\mathcal{T}$ may or may not exist.

We consider the cases where the user DL $\mathcal{L}_1$ is either the DL $\mathcal{EL}$ or where the user DL $\mathcal{L}_1$ is the DL $\mathcal{ALE}$ and the DL for the background terminology $\mathcal{L}_2$ is always the DL $\mathcal{ALC}$. In this chapter we show the following results:

- If $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox, then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions always exists.
- If $\mathcal{T}$ is a general $\mathcal{ALC}$-TBox allowing for general concept inclusion axioms (GCIs), then the lcs w.r.t. $\mathcal{T}$ of $\mathcal{ALE}(\mathcal{T})$-concept descriptions need not exist.
- We devise constructive computation algorithms to obtain *good* common subsumers of $\mathcal{ALE}(\mathcal{T})$-concept descriptions.

## 5.2 Existence of the lcs w.r.t. acyclic TBoxes

We first have to define the generalization of the usual lcs to the lcs w.r.t. (a)cyclic or general background TBoxes. Let $\mathcal{L}_1, \mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$, i.e., $\mathcal{L}_1$ allows for fewer concept constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, we call $\mathcal{L}_1(\mathcal{T})$-*concept descriptions* those $\mathcal{L}_1$-concept descriptions that may contain concepts defined in $\mathcal{T}$.

**Definition 52 (lcs w.r.t. a background TBox $\mathcal{T}$).** Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and the $\mathcal{L}_1(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, a $\mathcal{L}_1(\mathcal{T})$-concept description $D$ is the lcs of $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, iff

1. $C_i \sqsubseteq_{\mathcal{T}} D$ for all $1 \leq i \leq n$, and
2. if $E$ is an $\mathcal{L}_1(\mathcal{T})$-concept description satisfying $C_i \sqsubseteq_{\mathcal{T}} E$ for $i = 1, \ldots, n$, then $D \sqsubseteq_{\mathcal{T}} E$.

$\diamond$

The obtained concept description only uses concept constructors from $\mathcal{L}_1$, but it uses concept names defined in the $\mathcal{L}_2$-TBox. The existence of the lcs w.r.t. background terminologies has been first shown for the user DLs $\mathcal{EL}$ [BST04b] and then for $\mathcal{ALE}$ [BST04a; BST07], both, with respect to acyclic $\mathcal{ALC}$ background terminologies.

At first sight, one might assume that the existence of the lcs can be shown using results on approximation of DLs as introduced in Chapter 4. In fact, given an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and $\mathcal{EL}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$, one can first unfold $C_1, \ldots, C_n$ into $\mathcal{ALC}$-concept descriptions $C_1', \ldots, C_n'$, then build the $\mathcal{ALC}$-concept description $C := C_1' \sqcup \ldots \sqcup C_n'$, and finally approximate $C$ from above by an $\mathcal{EL}$-concept description $E$. However, $E$ then does not contain concept names defined in $\mathcal{T}$, and thus it is not necessarily the *least* $\mathcal{EL}(\mathcal{T})$-concept description subsuming $C_1, \ldots, C_n$. Let us illustrate this by a small example.

**Example 53.** *Assume that $\mathcal{L}_1$ is the DL $\mathcal{EL}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. Consider the $\mathcal{ALC}$-TBox*

$$\mathcal{T} := \{A \equiv P \sqcup Q\},$$

*and assume that we want to compute the lcs of the $\mathcal{EL}(\mathcal{T})$-concept descriptions $P$ and $Q$. Obviously, $A$ is the lcs of $P$ and $Q$ w.r.t. $\mathcal{T}$. If we were not allowed to use the name $A$ defined in $\mathcal{T}$, then the only common subsumer of $P$ and $Q$ in $\mathcal{EL}$ would be the top-concept $\top$.*

One might now assume that this effect can be overcome by computing a minimal rewriting of the concept descriptions w.r.t. the terminology [BKM00]. Recall from Section 2.4.3 that, roughly speaking, a minimal rewriting is obtained 're-introducing' names from the terminology for equivalent sub-concept descriptions. However, in Example 53, the concept description $E$ obtained using the approach based on approximation sketched above is $\top$, and this concept cannot be rewritten using the TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$.

Thus, to obtain any kind of common subsumer is rather simple from a conceptual point of view for $\mathcal{EL}(\mathcal{T})$- or $\mathcal{ALE}(\mathcal{T})$-concept descriptions, but the existence of the least common subsumer is not obvious for each of the two user DLs. In the following we give a proof for the existence for both of them.

### 5.2.1 Existence of the $\mathcal{EL}(\mathcal{T})$-lcs

We assume that $\mathcal{L}_1$ is $\mathcal{EL}$ and $\mathcal{L}_2$ is $\mathcal{ALC}$. In addition, we assume that the sets of concept and role names available for building concept descriptions are finite. We consider the case of acyclic TBoxes.

**Theorem 54.** *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox. The $\mathsf{lcs}_{\mathcal{EL}(\mathcal{T})}$ of $\mathcal{EL}(\mathcal{T})$-concept descriptions w.r.t. $\mathcal{T}$ always exists and can effectively be computed.*

This theorem is an easy consequence of the following facts:

1. If $D$ is an $\mathcal{EL}(\mathcal{T})$-concept description of role depth $k$, then there are (not necessarily distinct) roles $r_1, \ldots, r_k$ such that $D \sqsubseteq \exists r_1.\exists r_2. \ldots \exists r_k.\top$.

2. Let $C$ be an $\mathcal{EL}(\mathcal{T})$-concept description, and assume that the $\mathcal{ALC}$-concept description $C'$ obtained by unfolding $C$ w.r.t. $\mathcal{T}$ is satisfiable and has the role depth $\ell < k$. Then $C' \not\sqsubseteq \exists r_1.\exists r_2. \ldots \exists r_k.\top$, and thus $C \not\sqsubseteq_{\mathcal{T}} \exists r_1.\exists r_2. \ldots \exists r_k.\top$. In fact, the standard tableau-based algorithm for $\mathcal{ALC}$ applied to $C'$ constructs a tree-shaped interpretation of depth at most $\ell$ whose root individual belongs to $C'$, but not to $\exists r_1.\exists r_2. \ldots \exists r_k.\top$.

3. For a given bound $k$ on the role depth, there is only a finite number of inequivalent $\mathcal{EL}$-concept descriptions of role depth at most $k$. This is a consequence of the fact that we have assumed that the sets of concept and role names are finite, and it can be shown by induction on $k$.

To show that these facts imply Theorem 54, consider the $\mathcal{EL}(\mathcal{T})$-concept descriptions $C_1, \ldots, C_n$. If all of them are unsatisfiable w.r.t. $\mathcal{T}$, then one of them (e.g., $C_1$) can be taken as their lcs w.r.t. $\mathcal{T}$. Otherwise, assume that $C_i$ is satisfiable w.r.t. $\mathcal{T}$. Let $C_i'$ be the $\mathcal{ALC}$-concept description obtained by unfolding $C_i$ w.r.t. $\mathcal{T}$, and assume that its role depth is $\ell$. Now, take an arbitrary $\mathcal{EL}(\mathcal{T})$-concept description $E$ that is a common subsumer of $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$. Then, the role depth of $E$ is at most $\ell$. Otherwise, $C_i \sqsubseteq_{\mathcal{T}} E$ would be in contradiction to the facts 1. and 2. above. Thus, fact 3. implies that, up to equivalence, there are only finitely many common subsumers of $C_1, \ldots, C_n$ in $\mathcal{EL}(\mathcal{T})$. The least common subsumer is simply the conjunction of these finitely many $\mathcal{EL}(\mathcal{T})$-concept descriptions.

It is not hard to see that the above proof is effective in the sense that one can effectively compute (representatives of the equivalence classes of) all common subsumers of $C_1, \ldots, C_n$, and then build their conjunction. However, this brute-force algorithm is probably not useful in practice and a constructive method to obtain the lcs remains to be devised.

### 5.2.2 Existence of the $\mathcal{ALE}(\mathcal{T})$-lcs

Now we extend the user DL by the concept constructors atomic negation and value restrictions and prove the following theorem.

**Theorem 55.** *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox. The lcs of $\mathcal{ALE}(\mathcal{T})$-concept descriptions w.r.t. $\mathcal{T}$ always exists and can effectively be computed.*

Since the $n$-ary lcs can be obtained by iterating the application of the binary lcs, it is sufficient to show the theorem for the case where we want to build the lcs of two $\mathcal{ALE}(\mathcal{T})$-concept descriptions. To show the theorem in this case, we first need to show two propositions. Recall that we assume that the sets of concept and role names available for building concept descriptions are finite.

**Proposition 56.** *For a given bound $k$ on the role depth, there is only a finite number of inequivalent $\mathcal{ALE}$-concept descriptions of role depth at most $k$.*

This is a consequence of the fact that we have assumed that the sets of concept and role names are finite, and can easily be shown by induction on $k$.[19]

Given this lemma, a first attempt to show Theorem 55 could be to proceed as in the case of $\mathcal{EL}(\mathcal{T})$: Let $C_1, C_2$ be $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and assume that the role depths of the $\mathcal{ALC}$-concept description $C_1', C_2'$ obtained by unfolding the descriptions $C_i$ w.r.t. $\mathcal{T}$ are bounded by $k$. If we could show that this implies that the role depth of any common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$ is also bounded by $k$, then we could again obtain the lcs by simply building the (up to equivalence) finite conjunction of all common subsumers of $C_1, C_2$ in $\mathcal{ALE}(\mathcal{T})$.

However, due to the fact that in $\mathcal{ALC}$ and $\mathcal{ALE}$ we can define unsatisfiable concepts, this simple, role depth-based $\mathcal{EL}(\mathcal{T})$ approach does not work. In fact, $\bot$ has role depth

[19] In fact, this is a well-known result, which holds even for full first-order predicate logic formulae of bounded quantifier depth over a finite vocabulary.

0, but it is subsumed by any concept description. Preventing this problem by testing for unsatisfiable input concept descriptions is not enough due to the presence of value restrictions. For example, $\forall r.\bot$ is subsumed by $\forall r.F$ for arbitrary $\mathcal{ALE}(\mathcal{T})$-concept descriptions $F$, and thus the role depth of common subsumers cannot be bounded. However, we can show that common subsumers having a large role depth are too general anyway.

Before giving a more formal statement of this result in Proposition 61, we show some basic model-theoretic facts about $\mathcal{ALE}$ and $\mathcal{ALC}$, which will be employed in the proof of this proposition. An interpretation $\mathcal{I}$ is *tree-shaped* if the role relationships in $\mathcal{I}$ form a tree, i.e., if the directed graph $G_\mathcal{I} = (V_\mathcal{I}, E_\mathcal{I})$ with $V_\mathcal{I} = \Delta^\mathcal{I}$ and

$$E_\mathcal{I} = \{(d, d') \mid (d, d') \in r^\mathcal{I} \text{ for some role } r \in N_R\}$$

is a tree. An interpretation $\mathcal{I}$ is a *tree-shaped counterexample* to the subsumption question $C \sqsubseteq^?_\mathcal{T} D$ iff $\mathcal{I}$ is a tree-shaped model of $\mathcal{T}$ with root $d_0 \in C^\mathcal{I} \setminus D^\mathcal{I}$.

**Lemma 57.** *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox and $C, D$ be $\mathcal{ALC}$-concept descriptions. If $C \not\sqsubseteq_\mathcal{T} D$, then the subsumption question $C \sqsubseteq^?_\mathcal{T} D$ has a tree-shaped counterexample.*

**Proof.** Assume that $C \not\sqsubseteq_\mathcal{T} D$, and let $C', D'$ be the $\mathcal{ALC}$-concept descriptions obtained by unfolding $C, D$ w.r.t. $\mathcal{T}$. Then $C' \sqcap \neg D'$ is satisfiable. It is well-known that the tableau-based satisfiability procedure for $\mathcal{ALC}$ [SS91] then produces a tree-shaped interpretation $\mathcal{I}$ whose root $d_0$ satisfies $d_0 \in C'^\mathcal{I} \setminus D'^\mathcal{I}$. Since $C', D'$ do not contain concept names defined in $\mathcal{T}$, and since $\mathcal{T}$ is acyclic, we can assume w.l.o.g. that $\mathcal{I}$ is a model of $\mathcal{T}$. In fact, otherwise we can modify $\mathcal{I}$ by setting $A^\mathcal{I} := C'^\mathcal{I}_A$ for all defined concepts $A$, where $A \equiv C_A$ is the definition of $A$ in $\mathcal{T}$, and $C'_A$ is the unfolded concept description of $C_A$ w.r.t. $\mathcal{T}$. ❑

In case $D = \bot$, the statement $C \not\sqsubseteq_\mathcal{T} D$ is equivalent to saying that $C$ is satisfiable w.r.t. $\mathcal{T}$, and thus the lemma also implies that any $\mathcal{ALC}$-concept description that is satisfiable w.r.t. $\mathcal{T}$ has a tree-shaped model, i.e., a tree-shaped model of $\mathcal{T}$ with root $d_0 \in C^\mathcal{I}$. Of course, this and the above lemma also hold when the TBox is empty, i.e., for satisfiability and subsumption of concept descriptions.

Let $\mathcal{I}$ be a tree-shaped model of the acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$, and $C_0$ be an $\mathcal{ALC}$-concept description. An element $d$ of $\mathcal{I}$ is at *level* $k$ if the unique path from the root $d_0$ of $\mathcal{I}$ to $d$ has length $k$. A subdescription $F$ of $C_0$ is at *level* $k$ if it occurs within $k$ nestings of value and existential restrictions.

When evaluating $C_0$ in $\mathcal{I}$, i.e., when checking whether the root $d_0$ of $\mathcal{I}$ belongs to $C_0^\mathcal{I}$, we can directly use the inductive definition of the semantics of $\mathcal{ALC}$-concept descriptions. During this evaluation process, one recursively checks whether certain elements $d$ of $\mathcal{I}$ belong to $F^\mathcal{I}$ for subdescriptions $F$ of $C_0$. It is easy to see that, in such a recursive test, the level of $F$ in $C_0$ always coincides with the level of $d$ in $\mathcal{I}$. In particular, this means that elements of $\mathcal{I}$ that are at a level higher than the role depth of $C_0$ are irrelevant when evaluating $C_0$. The following lemma is an immediate consequence of this observation.

**Lemma 58.** *Let $C_0$ be an $\mathcal{ALC}$-concept description of role depth $\ell$, and let $\mathcal{I}, \mathcal{I}'$ be tree-shaped interpretations that differ from each other only on elements at levels larger than $\ell$. Then $d_0 \in C_0^\mathcal{I}$ iff $d_0 \in C_0^{\mathcal{I}'}$, where $d_0$ is the (common) root of $\mathcal{I}$ and $\mathcal{I}'$.*

In the proof of Proposition 61 we will need a specific result regarding the evaluation of $\mathcal{ALC}$-concept descriptions that are obtained by unfolding $\mathcal{ALE}(\mathcal{T})$-concept descriptions, where $\mathcal{T}$ is an acyclic $\mathcal{ALC}$-TBox. Before we can formulate this result in Lemma 59, we must introduce some more notation.

Let $C_0$ be an $\mathcal{ALC}$-concept description. We define under which conditions a subdescription $F$ of $C_0$ *occurs conjunctively in* $C_0$ by induction on the role level $\ell$ of $F$ in $C_0$:[20]

- if $\ell = 0$, then $C_0$ must be of the form $F_0 \sqcap F$;
- if $\ell > 0$, then $C_0$ must be of the form $F_0 \sqcap \exists r.C'$ or $F_0 \sqcap \forall r.C'$, where $F$ occurs conjunctively in $C'$ on role level $\ell - 1$.

The following lemma, which can easily be proved by induction on $\ell$, links this notion to $\mathcal{ALE}(\mathcal{T})$-concept descriptions. Given an acyclic $\mathcal{ALC}$-TBox $\mathcal{T}$ and an $\mathcal{ALE}(\mathcal{T})$-concept description $C_0$, the subdescription $F$ of $C_0$ is called *positive* if it is not a concept name that occurs within an atomic negation. For example, in the concept description $C_0 = \neg A \sqcap \exists r.\neg B$, the subdescriptions $A$ and $B$ are not positive, but all other subdescriptions (e.g., $\neg A$ or $\exists r.\neg B$) are positive.

**Lemma 59.** *Let $\mathcal{T}$ be an acyclic $\mathcal{ALC}$-TBox, and $C_0$ an $\mathcal{ALE}(\mathcal{T})$-concept description that contains the positive subdescription $F$ at some level $\ell$. In addition, let $C_0', F'$ be the $\mathcal{ALC}$-concept descriptions obtained by unfolding $C_0, F$ w.r.t. $\mathcal{T}$. Then $F'$ occurs conjunctively in $C_0'$ on level $\ell$.*

This lemma will be used to show that the next lemma is applicable in the proof of Proposition 61.

Let $C_0$ be an $\mathcal{ALC}$-concept description that contains the subdescription $F$ at some level $\ell \geq 0$ conjunctively, and let $\mathcal{I}$ be a tree-shaped interpretation with root $d_0$ such that $d_0 \in C_0^{\mathcal{I}}$. We modify $C_0$ into a new $\mathcal{ALC}$-concept description $C_\bot$ by replacing the subdescription $F$ by $\bot$. Now, assume that

- this replacement changes the evaluation of the concept description in $\mathcal{I}$, i.e., $d_0 \notin C_\bot^{\mathcal{I}}$.
- $\neg F$ is satisfiable, and thus there is a tree-shaped interpretation $\mathcal{J}$ with root $e_0$ such that $e_0 \notin F^{\mathcal{J}}$.

Without loss of generality we may assume that the domains of $\mathcal{I}$ and $\mathcal{J}$ are disjoint.

**Lemma 60.** *Let $C_0$ and $\mathcal{I}$ satisfy the properties stated above. Then there is a tree-shaped interpretation $\mathcal{I}'$ with root $d_0$ that differs from $\mathcal{I}$ only on elements at levels $\geq \ell$ such that $d_0 \notin C_0^{\mathcal{I}'}$.*

**Proof.** We prove the lemma by induction on $\ell$.
*Base case:* $\ell = 0$. In this case, $C_0$ is of the form $F_0 \sqcap F$. Let $\mathcal{I}'$ be a renamed copy

[20] The representation of $C_0$ is meant modulo associativity and commutativity of conjunction, and the fact that $\top$ is a unit for conjunction.

of $\mathcal{J}$, whose root has the name $d_0$ instead of $e_0$. Obviously, $e_0 \notin F^{\mathcal{J}}$ then implies $d_0 \notin F^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$.
*Induction step:* $\ell > 0$. In this case, $C_0$ is of the form $F_0 \sqcap \exists r.C'$ or $F_0 \sqcap \forall r.C'$, where $F$ is a conjunctive subdescription of $C'$ at level $\ell - 1$. Consequently, $C_\bot$ is of the form $F_0 \sqcap \exists r.C'_\bot$ or $F_0 \sqcap \forall r.C'_\bot$, where $C'_\bot$ is obtained from $C'$ by replacing the subdescription $F$ at level $\ell - 1$ by $\bot$.

First, consider the case where $C_0 = F_0 \sqcap \exists r.C'$ and $C_\bot = F_0 \sqcap \exists r.C'_\bot$. Obviously, $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin C_\bot^{\mathcal{I}}$ imply that $d_0 \in (\exists r.C')^{\mathcal{I}}$ and $d_0 \notin (\exists r.C'_\bot)^{\mathcal{I}}$. Let $d_1, \ldots, d_m$ be all the elements of $\mathcal{I}$ that satisfy $(d_0, d_i) \in r^{\mathcal{I}}$ and $d_i \in C'^{\mathcal{I}}$. Now, $d_0 \notin (\exists r.C'_\bot)^{\mathcal{I}}$ implies, for $i = 1, \ldots, m$, that $d_i \notin C'^{\mathcal{I}}_\bot$. Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be the tree-shaped interpretations obtained by taking the respective subtrees of $\mathcal{I}$ with roots $d_1, \ldots, d_m$. For $i = 1, \ldots, m$, we then have $d_i \in C'^{\mathcal{I}_i}$ and $d_i \notin C'^{\mathcal{I}_i}_\bot$. Since $F$ occurs conjunctively at level $\ell - 1$ in $C'$, the induction hypothesis yields a tree-shaped interpretation $\mathcal{I}'_i$ with root $d_i$ that differs from $\mathcal{I}_i$ only on elements at levels $\geq \ell - 1$, and such that $d_i \notin C'^{\mathcal{I}'_i}$.

The interpretation $\mathcal{I}'$ is obtained from $\mathcal{I}$ by replacing, for $i = 1, \ldots, m$, the subtree $\mathcal{I}_i$ with root $d_i$ by $\mathcal{I}'_i$. Obviously, $\mathcal{I}$ is tree-shaped and it differs from $\mathcal{I}'$ only on elements at levels $\geq \ell$. We claim that $d_0 \notin (\exists r.C')^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$. In fact, let $d$ be such that $(d_0, d) \in r^{\mathcal{I}'}$. By the definition of $\mathcal{I}'$, this implies that $(d_0, d) \in r^{\mathcal{I}}$. If $d = d_i$ for some $i, 1 \leq i \leq m$, then $d = d_i \notin C'^{\mathcal{I}'_i}$, and thus $d = d_i \notin C'^{\mathcal{I}'}$ since the subtree with root $d_i$ of $\mathcal{I}'$ coincides with $\mathcal{I}'_i$. Otherwise, $d \notin C'^{\mathcal{I}}$, and thus $d \notin C'^{\mathcal{I}'}$ since $\mathcal{I}$ coincides with $\mathcal{I}'$ on the respective subtrees with root $d$.

Second, consider the case where $C_0 = F_0 \sqcap \forall r.C'$ and $C_\bot = F_0 \sqcap \forall r.C'_\bot$. Obviously, $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin C_\bot^{\mathcal{I}}$ imply that $d_0 \in (\forall r.C')^{\mathcal{I}}$ and $d_0 \notin (\forall r.C'_\bot)^{\mathcal{I}}$. Let $d_1, \ldots, d_m$ be all the elements of $\mathcal{I}$ that satisfy $(d_0, d_i) \in r^{\mathcal{I}}$. Now, $d_0 \in (\forall r.C')^{\mathcal{I}}$ implies $d_i \in C'^{\mathcal{I}}$ for all $i, 1 \leq i \leq m$. In addition, $d_0 \notin (\forall r.C'_\bot)^{\mathcal{I}}$ implies that there exists a $j, 1 \leq j \leq m$, such that $d_j \notin C'^{\mathcal{I}}_\bot$. Let $\mathcal{I}_j$ be the tree-shaped interpretation obtained by taking the subtree of $\mathcal{I}$ with root $d_j$. Then, we have $d_j \in C'^{\mathcal{I}_j}$ and $d_j \notin C'^{\mathcal{I}_j}_\bot$. Since $F$ occurs conjunctively at level $\ell - 1$ in $C'$, the induction hypothesis yields a tree-shaped interpretation $\mathcal{I}'_j$ with root $d_j$ that differs from $\mathcal{I}_j$ only on elements at levels $\geq \ell - 1$, and such that $d_j \notin C'^{\mathcal{I}'_j}$.

The interpretation $\mathcal{I}'$ is obtained from $\mathcal{I}$ be replacing the subtree $\mathcal{I}_j$ with root $d_j$ by $\mathcal{I}'_j$. Obviously, $\mathcal{I}$ is tree-shaped and it differs from $\mathcal{I}'$ only on elements at levels $\geq \ell$. We claim that $d_0 \notin (\forall r.C')^{\mathcal{I}'}$, and thus $d_0 \notin C_0^{\mathcal{I}'}$. This is an immediate consequence of the following two facts: (i) $(d_0, d_j) \in r^{\mathcal{I}'}$, and (ii) $d_j \notin C'^{\mathcal{I}'_j}$, and thus $d_j \notin C'^{\mathcal{I}'}$ since the subtree with root $d_j$ of $\mathcal{I}'$ coincides with $\mathcal{I}'_j$. ❑

We are now ready to prove the key proposition, which gives a bound of the role depth of common subsumers in relation to the maximal role depth of the input concept descriptions.

**Proposition 61.** *Let $C_1, C_2$ be $\mathcal{ALE}(\mathcal{T})$-concept descriptions that are both satisfiable w.r.t. $\mathcal{T}$, and assume that the role depths of the $\mathcal{ALC}$-concept descriptions $C'_1, C'_2$ obtained by unfolding the descriptions $C_1, C_2$ w.r.t. $\mathcal{T}$ are bounded by $k$. If the $\mathcal{ALE}(\mathcal{T})$-concept description $D$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$, then there is an $\mathcal{ALE}(\mathcal{T})$-concept description $D_0 \sqsubseteq_{\mathcal{T}} D$ of role depth at most $k + 1$ that is also a*

*common subsumer of* $C_1, C_2$ *w.r.t.* $\mathcal{T}$.

**Proof.** Let $D$ be an $\mathcal{ALE}(\mathcal{T})$-concept description that is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$. If the role depth of $D$ is bounded by $k+1$, then we are done since we can take $D_0 = D$. Otherwise, $D$ contains at least one subdescription on level $k+1$ that is an existential or a value restriction. Choose such a subdescription $F$. Obviously, $F$ is positive. We modify $D$ into a concept description $\widehat{D}$ as follows. We replace $F$ by either $\top$ or $\bot$:

- if $F$ is equivalent to $\top$ w.r.t. $\mathcal{T}$, then it is replaced by $\top$;
- otherwise, $F$ is replaced by $\bot$.

Since $F$ is a positive subdescription of $E$ and all the concept constructors other than atomic negation available in $\mathcal{ALE}$ are monotonic, it is clear that $\widehat{D} \sqsubseteq_{\mathcal{T}} D$. It remains to be shown that $\widehat{D}$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$. In fact, once we have shown this, we can obtain $D_0$ by applying this construction until all subdescriptions at level $k+1$ that are existential or a value restrictions are replaced by either $\top$ or $\bot$. Obviously, the resulting description $D_0$ has role depth at most $k+1$ and satisfies $D_0 \sqsubseteq_{\mathcal{T}} D$.

If $F$ was replaced by $\top$, then $F \equiv_{\mathcal{T}} \top$, and thus $\widehat{D} \equiv_{\mathcal{T}} D$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$. Thus, assume that $F$ was replaced by $\bot$. To show that also in this case $\widehat{D}$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$, we assume to the contrary that $C_i \not\sqsubseteq_{\mathcal{T}} \widehat{D}$ for $i = 1$ or $i = 2$. We show that this assumption leads to a contradiction.

Let $D', \widehat{D}', F'$ be the $\mathcal{ALC}$-concept descriptions obtained by respectively unfolding $D, \widehat{D}, F$. By Lemma 59, $F'$ is a subdescription of $D'$ that occurs conjunctively in $D'$ at level $k+1$. In addition, since $F$ was replaced by $\bot$, $F$ is not equivalent to $\top$ w.r.t. $\mathcal{T}$, and thus $\neg F'$ is satisfiable. Since $C_i \not\sqsubseteq_{\mathcal{T}} \widehat{D}$, we know that $C_i' \not\sqsubseteq \widehat{D}'$, and thus there is a tree-shaped interpretation $\mathcal{I}$ such that the root $d_0$ of this tree belongs to $C_i'^{\mathcal{I}}$, but not to $\widehat{D}'^{\mathcal{I}}$. Since $C_i \sqsubseteq_{\mathcal{T}} D$, we also know that $C_i' \sqsubseteq D'$, and thus $d_0 \in D'^{\mathcal{I}}$.

Now, $d_0 \notin \widehat{D}'^{\mathcal{I}}$ and $d_0 \in D'^{\mathcal{I}}$ together with the satisfiability of $\neg F'$ and the way $\widehat{D}$ was constructed from $D$ imply that Lemma 60 is applicable. Thus, there is a tree-shaped interpretation $\mathcal{I}'$ with root $d_0$ that differs from $\mathcal{I}$ only on elements at levels $\geq k+1$, and such that $d_0 \notin D'^{\mathcal{I}'}$.

Since a change of the interpretation at a level larger than $k$ does not influence the evaluation of a concept description of depth at most $k$ (see Lemma 58), $d_0 \in C_i'^{\mathcal{I}}$ implies $d_0 \in C_i'^{\mathcal{I}'}$. However, since $C_i \sqsubseteq_{\mathcal{T}} D$ yields $C_i' \sqsubseteq D'$, this implies $d_0 \in D'^{\mathcal{I}'}$, which yields the desired contradiction. ❑

Theorem 54 is now an immediate consequence of Proposition 56 and Proposition 61. In fact, to compute the lcs of $C_1, C_2$ w.r.t. $\mathcal{T}$, it is enough to compute the (up to equivalence) finite set of all $\mathcal{ALE}(\mathcal{T})$-concept descriptions of role depth at most $k+1$, check which of them are common subsumers of $C_1, C_2$ w.r.t. $\mathcal{T}$, and then build the conjunction $E$ of these common subsumers. Proposition 56 ensures that the conjunction is finite. By definition, $E$ is a common subsumer of $C_1, C_2$ w.r.t. $\mathcal{T}$, and Proposition 61 ensures that for any common subsumer $D$ of $C_1, C_2$ w.r.t. $\mathcal{T}$, there is a conjunct $D_0$ in $E$ such that $D_0 \sqsubseteq_{\mathcal{T}} D$, and thus $E \sqsubseteq_{\mathcal{T}} D$.

This generate and test-based brute-force method to obtain the $\mathcal{ALE}(\mathcal{T})$-lcs is not very useful in practice. Next, we examine the existence of the lcs in the case where the background TBox is not restricted to be unfoldable.

## 5.3 Non-existence of the lcs w.r.t. general TBoxes

In this section we show that the lcs for general TBoxes does neither exist for $\mathcal{EL}(\mathcal{T})$- nor for $\mathcal{ALE}(\mathcal{T})$-TBoxes. We start with $\mathcal{EL}(\mathcal{T})$ and consider the case of general TBoxes.

**Theorem 62.** *Let $\mathcal{T} := \{A \sqsubseteq \exists r.A,\ B \sqsubseteq \exists r.B\}$. Then, the lcs of the $\mathcal{EL}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

**Proof.** Let $E_n$ denote the $\mathcal{EL}$-concept description $\exists r.\exists r \cdots \exists r.\top$ of role depth $n$. For all $n \geq 0$, $E_n$ is a common subsumer of $A$ and $B$ w.r.t. $\mathcal{T}$. Assume that $D$ is a lcs of $A$ and $B$, and let $\ell$ be the role depth of $D$. If $D$ contains neither $A$ nor $B$, then $D \not\sqsubseteq_{\mathcal{T}} E_n$ for all $n > \ell$, which is a contradiction. However, if $D$ contains $A$, then it is easy to see that $D$ cannot be a subsumer of $B$, and if $D$ contains $B$, then it cannot be a subsumer of $A$. Consequently, such a lcs $D$ cannot exist. ❑

Note that this example is very similar to the one showing non-existence of the lcs in $\mathcal{EL}$ with cyclic terminologies interpreted with descriptive semantics [Baa03a]. However, the proof of the result in [Baa03a] is more involved since there one is allowed to extend the terminology in order to build the lcs. However, the same example can be used to prove that the lcs for $\mathcal{ALE}(\mathcal{T})$-concept description does not need to exist.

**Theorem 63.** *Let $\mathcal{T} := \{A \sqsubseteq \exists r.A,\ B \sqsubseteq \exists r.B\}$, where $A, B$ are distinct concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

**Proof.** Consider a common subsumer $E$ of $A, B$ w.r.t. $\mathcal{T}$. Without loss of generality we can assume that the $\mathcal{ALE}(\mathcal{T})$-concept description $E$ is a conjunction of (negated) concept names, value restrictions, and existential restrictions. We claim that this conjunction can actually only contain existential restrictions for the role $r$.

Assume that the concept name $P$ is contained in this conjunction. We restrict our attention to the case where $P \neq C \in \{A, B\}$. Consider the interpretation $\mathcal{I}$ that consists of one element $c$, which belongs to $C$ and to no other concept name, and which is related to itself via the role $r$. Then $\mathcal{I}$ is a model of $\mathcal{T}$, and $c \in C^{\mathcal{I}}$. However, $c \notin P^{\mathcal{I}}$, which is a contradiction since $P$ occurs in the top-level conjunction of $E$, and we have assumed that $C \sqsubseteq_{\mathcal{T}} E$. Similarly, we can show that no negated concept name can occur in this conjunction.

For similar reasons, the conjunction cannot contain a value restriction $\forall s.F$ where $F \not\equiv_{\mathcal{T}} \top$. In fact, if $F \not\equiv_{\mathcal{T}} \top$, then there is a model $\mathcal{I}_{\neg F}$ of $\mathcal{T}$ that contains an element $d_0$ with $d_0 \notin F^{\mathcal{I}_{\neg F}}$. We extend $\mathcal{I}_{\neg F}$ to an interpretation $\mathcal{I}$ by adding a new element $a$, which belongs to $A$ and to no other concept name, and which is related to itself via the role $r$, and to $d_0$ via the role $s$. Then $\mathcal{I}$ is a model of $\mathcal{T}$, and $a \in A^{\mathcal{I}}$. However, $a \notin (\forall s.F)^{\mathcal{I}}$, which is a contradiction since $A \sqsubseteq_{\mathcal{T}} E$.

Thus, we may assume w.l.o.g. that both the conjunction of (negated) concept names and the conjunction of value restrictions is empty. Now, consider an existential

restriction $\exists s.F$. By using a construction similar to the ones above, we can show that $s$ must be equal to $r$, i.e., we have an existential restriction of the form $\exists r.F$. Otherwise, we assume w.l.o.g. that $A \not\sqsubseteq_{\mathcal{T}} F$, i.e., there is a model $\mathcal{I}_0$ of $\mathcal{T}$ that contains an element $d_0$ s.t. $d_0 \in A^{\mathcal{I}_0} \setminus F^{\mathcal{I}_0}$. This is a contradiction to $A \sqsubseteq_{\mathcal{T}} E \sqsubseteq_{\mathcal{T}} \exists r.F$ since using $\mathcal{I}_0$ we can easily construct a model $\mathcal{I}$ of $\mathcal{T}$ that contains an element $a$ that belongs to $A$, but not to $\exists r.F$. In fact, $\mathcal{I}$ is obtained from $\mathcal{I}_0$ by adding a new element $a$, which belongs to $A$ and to no other concept name, and which is related to $d_0$ via the role $r$.

We can now apply induction over the role depth of the common subsumer $E$ of $A, B$ to show that $E$ is equivalent w.r.t. $\mathcal{T}$ to an $\mathcal{ALE}$-concept description from the following set of descriptions: $\mathcal{S}$ is the smallest set of $\mathcal{ALE}$-concept descriptions such that (i) $\top$ belongs to $\mathcal{S}$, (ii) $\mathcal{S}$ is closed under conjunction and (iii) if $F$ belongs to $\mathcal{S}$, then $\exists r.F$ also belongs to $\mathcal{S}$. Conversely, it is easy to show (by induction on the size of elements of $\mathcal{S}$) that any element of $\mathcal{S}$ is a common subsumer of $A, B$ w.r.t. $\mathcal{T}$.

Thus, a least common subsumer of $A, B$ w.r.t. $\mathcal{T}$ must be a least element of $\mathcal{S}$. Since the elements of $\mathcal{S}$ do not contain $A, B$, here least has to be understood w.r.t. subsumption of concept descriptions (i.e., without a TBox). However, $\mathcal{S}$ does not have a least element w.r.t. subsumption. On the one hand, $\mathcal{S}$ obviously contains elements of arbitrary role depth. On the other hand, an element $D$ of role depth $\ell$ cannot be subsumed by an element $E$ of role depth $k > \ell$: if $d \in E^{\mathcal{I}}$ for some interpretation $\mathcal{I}$ and element $d$ of $\mathcal{I}$, then there is a path of length $k$ starting from $d$ in the graph $G_{\mathcal{I}}$; in contrast, there is an interpretation $\mathcal{I}_0$ and an element $d_0$ of $\mathcal{I}_0$ such that $d_0 \in D^{\mathcal{I}_0}$, but all paths in $G_{\mathcal{I}_0}$ starting with $d_0$ have length $\leq \ell < k$. ❑

It is easy to see that the same proof also works for the cyclic TBox $\mathcal{T} := \{A \equiv \exists r.A, B \equiv \exists r.B\}$.

**Corollary 64.** *Let $\mathcal{T} := \{A \equiv \exists r.A,\ B \equiv \exists r.B\}$, where $A, B$ are distinct concept names. Then, the lcs of the $\mathcal{ALE}(\mathcal{T})$-concept descriptions $A, B$ w.r.t. $\mathcal{T}$ does not exist.*

The results from this and the previous subsection tell us for which background TBox formalisms we can hope for to be able to devise lcs computation algorithms. While this is possible for unfoldable TBoxes as a background ontology, is it not for cyclic or general ones. The brute force algorithm for obtaining the lcs for $\mathcal{ALE}(\mathcal{T})$ is hardly useful in practice.

## 5.4 Approximative approaches: good common subsumers

The brute-force algorithm for computing the lcs in $\mathcal{ALE}(\mathcal{T})$ w.r.t. an acyclic background $\mathcal{ALC}$-TBox by generating all subsumers up to a certain role depth and then conjoining them is not useful in practice since the number of concept descriptions that must be considered is very large (super-exponential in the role depth of the input concept descriptions). In addition, w.r.t. cyclic or general TBoxes the lcs need not exist. In the bottom-up construction of DL knowledge bases, it is not really necessary to take the *least* common subsumer, since using it may even result in over-fitting. A common subsumer that is not too general can also be used for our purpose. In this section, we introduce several approaches for computing such 'good' common subsumers w.r.t. a background TBox.

### 5.4.1 A good common subsumer in $\mathcal{ALE}$ w.r.t. a background TBox

Let $\mathcal{T}$ be a background TBox in some DL $\mathcal{L}_2$ extending $\mathcal{ALE}$ such that subsumption in $\mathcal{L}_2$ w.r.t. this kind of TBoxes is decidable.[21] Let $C, D$ be normalized $\mathcal{ALE}(\mathcal{T})$-concept descriptions and we are interested in their lcs. If we ignore the TBox, then we can simply apply the computation algorithm for the lcs of $\mathcal{ALE}$-concept descriptions (as displayed in Figure 3.2 on page 48) without background terminology to compute a common subsumer. However, in this context, taking

$$\bigsqcap_{A \in prim(C) \cap prim(D)} A$$

is not the best we can do. In fact, some of these concept names may be constrained by the TBox, and thus there may be relationships between them that we ignore by simply using the intersection. By $\mathsf{names}(C)$ we denote all concept names and by $\overline{\mathsf{names}}(C)$ we denote all negated names appearing on the top-level of $C$. Our proposal for modifying the $\mathcal{ALE}$-lcs algorithm is to replace the above conjunction by the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names and negated concept names that subsumes (w.r.t. $\mathcal{T}$) both

$$\bigsqcap_{A \in \mathsf{names}(C)} A \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C)} \neg B \quad \text{and} \quad \bigsqcap_{A' \in \mathsf{names}(D)} A' \sqcap \bigsqcap_{\neg B' \in \overline{\mathsf{names}}(D)} \neg B'.$$

We modify the lcs algorithm for $\mathcal{ALE}$ in this way, not only on the top-level of the input concepts, but also in the recursive steps. It is easy to show that the $\mathcal{ALE}(\mathcal{T})$-concept description computed by this modified algorithm still is a common subsumer of $A, B$ w.r.t. $\mathcal{T}$.

**Proposition 65.** *The $\mathcal{ALE}(\mathcal{T})$-concept description $E$ obtained by applying the modified lcs algorithm to $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C, D$ is a common subsumer of $C$ and $D$ w.r.t. $\mathcal{T}$, i.e., $C \sqsubseteq_{\mathcal{T}} E$ and $D \sqsubseteq_{\mathcal{T}} E$.*

In general, this common subsumer will be more specific than the one obtained by ignoring $\mathcal{T}$, though it need not be the least common subsumer. In the following, we will call the common subsumer computed this way *good common subsumer (gcs)*, and the algorithm that computes it the *gcs algorithm*.

**Example 66.** *As a simple example, consider the $\mathcal{ALC}$-TBox $\mathcal{T}$:*

$$\begin{aligned}
\mathsf{NoSon} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Female},\\
\mathsf{NoDaughter} &\equiv \forall \mathsf{has\text{-}child}.\neg\mathsf{Female},\\
\mathsf{SonRichDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Rich})),\\
\mathsf{DaughterHappyDoctor} &\equiv \forall \mathsf{has\text{-}child}.(\neg\mathsf{Female} \sqcup (\mathsf{Doctor} \sqcap \mathsf{Happy})),\\
\mathsf{ChildrenDoctor} &\equiv \forall \mathsf{has\text{-}child}.\mathsf{Doctor},
\end{aligned}$$

*and the $\mathcal{ALE}$-concept descriptions*

$$\begin{aligned}
C &:= \exists \mathsf{has\text{-}child}.(\mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}),\\
D &:= \exists \mathsf{has\text{-}child}.(\mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}).
\end{aligned}$$

[21] Note that the TBox $\mathcal{T}$ used as background terminology may now be a general TBox.

*By ignoring the TBox, we obtain the $\mathcal{ALE}(\mathcal{T})$-concept description $\exists$has-child.$\top$ as a common subsumer of $C, D$. However, if we take into account that both concept descriptions* NoSon $\sqcap$ DaughterHappyDoctor *and* NoDaughter $\sqcap$ SonRichDoctor *are subsumed by the concept* ChildrenDoctor*, then we obtain the more specific common subsumer* $\exists$has-child.ChildrenDoctor*. The gcs of $C, D$ is even more specific. In fact, the least conjunction of (negated) concept names subsuming both* NoSon$\sqcap$DaughterHappyDoctor *and* NoDaughter $\sqcap$ SonRichDoctor *is*

$$\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor},$$

*and thus the gcs of $C, D$ is*

$$\exists \mathsf{has\text{-}child}.(\mathsf{ChildrenDoctor} \sqcap \mathsf{DaughterHappyDoctor} \sqcap \mathsf{SonRichDoctor}).$$

*The conjunct* ChildrenDoctor *is actually redundant since it is implied by the remainder of the conjunction.*

In order to implement the gcs algorithm, we must be able to compute the smallest conjunction of (negated) concept names that subsumes two such conjunctions $C_1$ and $C_2$ w.r.t. $\mathcal{T}$ in an efficient way. In principle, one can compute this smallest conjunction by testing, for every (negated) concept name whether it subsumes both $C_1$ and $C_2$ w.r.t. $\mathcal{T}$, and then take the conjunction of those (negated) concept names for which the test was positive. However, this results in a large number of (possibly quite expensive) calls to the subsumption algorithm for $\mathcal{L}_2$ w.r.t. (general or (a)cyclic) TBoxes. Instead, we propose to use attribute exploration, a method from formal concept analysis (see [GW99]), for this purpose.

In order to apply attribute exploration to the task of computing the subsumption lattice[22] of conjunctions of (negated) concept names (some of which may be defined concepts in an $\mathcal{L}_2$-TBox $\mathcal{T}$), a formal context whose concept lattice is isomorphic to the subsumption lattice we are interested in was defined in [BST07]. A detailed discussion on how to employ attribute exploration in this context is beyond the scope of this thesis and is provided in the thesis by Sertkaya, see [Ser07]. However by attribute exploration we can compute the concept lattice of the conjunctions of (negated) concept names fast and, since in our application scenario the TBox $\mathcal{T}$ is assumed to be fixed, it is only necessary to compute it once in advance and reuse it later. Thus equipped with the concept lattice, we can obtain the supremum of the conjunction of (negated) concept names appearing in the input concepts and use it in the above described fashion to obtain a gcs of the input concept descriptions.

### 5.4.2 Using $\mathcal{ALE}$-unfolding when computing the gcs

If the background terminology is an acyclic TBox $\mathcal{T}$, then one can employ an appropriate partial unfolding of $\mathcal{T}$ in order to uncover $\mathcal{ALE}$-parts hidden within the defined

[22] In general, the subsumption relation induces a partial order, and not a lattice structure on concepts. However, in the case of conjunctions of (negated) concept names, all infima exist, and thus also all suprema.

concepts. The idea is that the gcs algorithm will possibly yield a more specific common subsumer if it can make use of these 'hidden' $\mathcal{ALE}$-concepts.

For instance, in Example 66, the concepts defining NoSon and NoDaughter are actually $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus $C, D$ can be unfolded to

$$\begin{aligned} C' &:= \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \mathsf{DaughterHappyDoctor}), \\ D' &:= \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\neg\mathsf{Female} \sqcap \mathsf{SonRichDoctor}), \end{aligned}$$

before computing their gcs. The two concepts defining DaughterHappyDoctor and SonRichDoctor are not $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and thus these two names cannot be unfolded. However, in this example, the common subsumer computed by applying the gcs algorithm to the unfolded concepts $C', D'$ is $\exists \mathsf{has\text{-}child}.\top$, which is actually less specific than the result of applying the gcs algorithm to the not unfolded concepts $C, D$. To overcome this problem, we do the partial unfolding, but also keep the defined concepts that we have unfolded. In the example, this yields the unfolded concepts

$$\begin{aligned} C'' &:= \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \mathsf{NoSon} \sqcap \mathsf{DaughterHappyDoctor}), \\ D'' &:= \exists \mathsf{has\text{-}child}.(\forall \mathsf{has\text{-}child}.\neg\mathsf{Female} \sqcap \mathsf{NoDaughter} \sqcap \mathsf{SonRichDoctor}). \end{aligned}$$

If we apply the gcs algorithm to $C'', D''$, then we obtain (up to equivalence w.r.t. $\mathcal{T}$) the same common subsumer as obtained from $C, D$, i.e., in this case the unfolding does not yield a more specific result.

However, it is easy to construct examples where this kind of unfolding leads to better results. For instance, if we apply the gcs algorithm to $\forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$ and $\mathsf{NoSon} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Happy}$, then the result is $\top$. In contrast, if we apply it to the unfolded concept descriptions $\forall \mathsf{has\text{-}child}.(\mathsf{Female} \sqcap \mathsf{Doctor})$ and $\mathsf{NoSon} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Female} \sqcap \forall \mathsf{has\text{-}child}.\mathsf{Happy}$, then the result is the more specific common subsumer $\forall \mathsf{has\text{-}child}.\mathsf{Female}$.

Before checking whether a defined concept should be unfolded, it is useful to transform it into NNF by applying the rules from Definition 34. For example, the concept description $\neg\forall \mathsf{has\text{-}child}.\mathsf{Female}$ is not an $\mathcal{ALE}$-concept description, but its NNF $\exists \mathsf{has\text{-}child}.\neg\mathsf{Female}$ is. More formally, we define the $\mathcal{ALE}$*-unfolding* of (negated) concept names defined in $\mathcal{T}$ and of $\mathcal{ALE}(\mathcal{T})$-concept descriptions as follows.

**Definition 67 ($\mathcal{ALE}$-unfolding).** Let $\mathcal{T}$ be an acyclic TBox, let $A$ be a concept name defined in $\mathcal{T}$, and let $A \equiv C$ be its definition. We first build the NNF $C'$ of $C$. If $C'$ is not an $\mathcal{ALE}(\mathcal{T})$-concept description, then the $\mathcal{ALE}$-unfolding of $A$ is $A$. Otherwise, it is $A \sqcap C''$, where $C''$ is obtained from $C'$ by replacing all (negated) defined concept names in $C'$ by their $\mathcal{ALE}$-unfolding. To obtain the $\mathcal{ALE}$-unfolding of $\neg A$, we just apply the same approach to $\neg C$. The $\mathcal{ALE}$-unfolding of an $\mathcal{ALE}(\mathcal{T})$-concept description is obtained by replacing all (negated) defined concept names by their $\mathcal{ALE}$-unfoldings. $\diamond$

Note that this recursive definition of an $\mathcal{ALE}$-unfolding is well-founded since the TBox is assumed to be acyclic. As an example, consider the TBox $\mathcal{T}$ consisting of

$$A \equiv \neg\forall r.(B_1 \sqcup B_2), \quad B_1 \equiv P \sqcup Q, \quad B_2 \equiv P \sqcap Q.$$

Then we obtain $A \sqcap \exists r.(\neg B_1 \sqcap \neg P \sqcap \neg Q \sqcap \neg B_2)$ as the $\mathcal{ALE}$-unfolding of $A$.

It is easy to see that $\mathcal{ALE}$-unfolding may lead to more specific common subsumers, but never to less specific ones.

**Proposition 68.** *Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox, $C, D$ $\mathcal{ALE}(\mathcal{T})$-concept descriptions with $\mathcal{ALE}$-unfolding $C', D'$, and let $E$ ($E'$) be the result of applying the gcs algorithm to $C, D$ ($C', D'$). Then $E'$ is a common subsumer of $C, D$ that is at least as good as $E$, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E' \sqsubseteq_{\mathcal{T}} E$.*

$\mathcal{ALE}$-unfolding can also be applied to cyclic $\mathcal{L}_2$-TBoxes, provided that the cycles involve only non-$\mathcal{ALE}$ parts of the TBox or termination of the unfolding is ensured otherwise. This is, for example, the case in the TBox $\mathcal{T} := \{A \equiv \exists r.B,\ B \equiv P \sqcup A\}$.

### 5.4.3 Alternative approaches for computing common subsumers

In Section 5.2 we have already sketched an approach based on approximation, which works if the TBox $\mathcal{T}$ is acyclic, $\mathcal{L}_2$ allows for disjunction, and one can compute the approximation of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions. For example, if we take $\mathcal{ALC}$ as $\mathcal{L}_2$, then all these conditions are satisfied.

**Definition 69 (Common subsumer by approximation (acs)).** Assume that $\mathcal{L}_2$ allows for disjunction, and that the approximation of $\mathcal{L}_2$-concept descriptions by $\mathcal{ALE}$-concept descriptions can be computed. Let $\mathcal{T}$ be an acyclic $\mathcal{L}_2$-TBox. Given $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C, D$, let $C', D'$ be the concept descriptions obtained by unfolding $C, D$ w.r.t. $\mathcal{T}$. Let $E$ be the $\mathcal{ALE}$-approximation of $C' \sqcup D'$, then $E$ is the *common subsumer by approximation (acs)* of $C, D$ w.r.t. $\mathcal{T}$. ◇

In Section 5.2, we have shown by Example 53 that the acs can be less specific than the lcs. In this example (Example 53), the gcs coincides with the lcs, and thus is also more specific than the acs: in fact, w.r.t. the TBox $\mathcal{T} = \{A \equiv P \sqcup Q\}$, the smallest conjunction of concept names above both $P$ and $Q$ is $A$, and thus the gcs of $P$ and $Q$ is $A$.

There are, however, also examples where the gcs is less specific than the acs. For instance, consider the TBox

$$\mathcal{T} = \{A \equiv \exists r.A_1 \sqcup \exists r.A_2, \quad B \equiv \exists r.B_1 \sqcup \exists r.B_2\}.$$

With respect to this TBox, the gcs of $A, B$ is $\top$, whereas the acs is the more specific common subsumer $\exists r.\top$.

The gcs algorithm makes use of the subsumption relationships between conjunctions of (negated) concept names. Usually, these relationships are not known for a given TBox, and thus we must either precompute them, for instance by attribute exploration, or compute them on the fly. Both may be quite expensive. What is usually known for a given TBox $\mathcal{T}$ are all subsumption relationships between the concept names occurring in $\mathcal{T}$. This information can be used as follows. Given two conjunctions

$$\bigsqcap_{A \in \mathsf{names}(C)} A \ \sqcap \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C)} \neg B \qquad \text{and} \qquad \bigsqcap_{A' \in \mathsf{names}(D)} A' \ \sqcap \bigsqcap_{\neg B' \in \overline{\mathsf{names}}(D)} \neg B',$$

the gcs algorithm takes the smallest (w.r.t. subsumption w.r.t. $\mathcal{T}$) conjunction of concept names and negated concept names that subsumes (w.r.t. $\mathcal{T}$) both conjunctions. In contrast, the algorithm that just ignores the TBox would take

$$\bigsqcap_{A \in \mathsf{names}(C) \cap \mathsf{names}(D)} A \quad \sqcap \quad \bigsqcap_{\neg B \in \overline{\mathsf{names}}(C) \cap \overline{\mathsf{names}}(D)} \neg B.$$

Using the subsumption relationships between concept names, we can come up with a new approach that lies between these two approaches.

**Definition 70 (Subsumption closure).** Let $\mathcal{T}$ be a TBox, and $S$ ($\overline{S}$) a set of (negated) concept names. The *subsumption closure* of $S$ ($\overline{S}$) w.r.t. $\mathcal{T}$ is a set of (negated) concept names, which is defined as follows:

$$\begin{array}{lcl} \mathsf{SC}(S) & := & \{A \mid \exists B \in S.\ B \sqsubseteq_{\mathcal{T}} A\}, \\ \mathsf{SC}(\overline{S}) & := & \{\neg A \mid \exists \neg B \in \overline{S}.\ A \sqsubseteq_{\mathcal{T}} B\}. \end{array}$$

◇

We can obtain a notion of common subsumer that yields more specific results than the acs based on the subsumption closure.

**Definition 71 (Common subsumer by subsumption closure (scs)).** We call the algorithm for computing common subsumers obtained by first building the subsumption closures, and then intersecting the closures, i.e., using

$$\bigsqcap_{A \in \mathsf{SC}(\mathsf{names}(C)) \cap \mathsf{SC}(\mathsf{names}(D))} A \quad \sqcap \quad \bigsqcap_{\neg B \in \mathsf{SC}(\overline{\mathsf{names}}(C)) \cap \mathsf{SC}(\overline{\mathsf{names}}(D))} \neg B.$$

instead of the intersection $prim(C) \cap prim(D)$ in the $\mathcal{ALE}$-lcs algorithm, the *scs algorithm*, and the result of applying it to $\mathcal{ALE}(\mathcal{T})$-concept descriptions $C, D$ the *scs of* $C, D$ *w.r.t.* $\mathcal{T}$. ◇

**Proposition 72.** *Let $\mathcal{T}$ be an $\mathcal{L}_2$-TBox, $C, D$ $\mathcal{ALE}(\mathcal{T})$-concept descriptions, and let $E$ ($E'$) be the result of applying the gcs (scs) algorithm to $C, D$. Then $E'$ is a common subsumer of $C, D$ that is at most as good as the gcs $E$, i.e., $C \sqsubseteq_{\mathcal{T}} E'$, $D \sqsubseteq_{\mathcal{T}} E'$, and $E \sqsubseteq_{\mathcal{T}} E'$.*

The claim is obvious, since the gcs constructs a conjunction that subsumes the conjunction of (negated) names appearing in the input concept descriptions, while the scs merely collects concept names from the TBox that subsume *one* (negated) concept name that appears in the input concepts.

The computation methods here proposed applicable for common subsumers are, of course, also useful for applications, where only one DL and one TBox is used. In this case the methods will yield concept descriptions at most as specific as the lcs in the general case.

**Extending computation of gcs, acs and scs to number restrictions.** Observe that the acs algorithm, as well as the two other methods to compute common subsumers proposed in this section, either use computation methods for NSIs in the user DL $\mathcal{ALE}$ or employ variants of them that only modify the way concept names are treated. In case of the acs, one can obviously also obtain a common subsumer in the presence of number restrictions in the background DL and, more importantly, in the user DL by using the computation of the $\mathcal{ALEN}$-approximations of the disjunction of the input concepts.

In case of the gcs and the scs, the $\mathcal{ALEN}$-lcs computation algorithm can be modified in the same fashion as we described it for the lcs computation algorithm for $\mathcal{ALE}$. These modifications yield computation methods for common subsumers of $\mathcal{ALEN}(\mathcal{T})$-concept descriptions w.r.t. $\mathcal{ALCN}$ background TBoxes.

In this chapter we have investigated the computation of common subsumers for the customization of background TBoxes. In this setting the common subsumers are expressed in a user DL not offering disjunction, while using concept names from the background TBox, which is written in a more expressive DL offering disjunction. We investigated the existence of least common subsumers in this setting. While for unfoldable TBoxes the lcs of $\mathcal{ALE}(\mathcal{T})$-concept descriptions exists, this is not the case if the background TBox is a general or cyclic one. Unfortunately, no constructive method to obtain the lcs for $\mathcal{ALE}(\mathcal{T})$ w.r.t. unfoldable $\mathcal{ALC}$-background TBoxes could be devised. Instead we proposed three methods to obtain good common subsumers, which are not necessarily the least ones (w.r.t. subsumption). These methods can be extended to handle number restrictions in a straightforward way.

Together with the approximation based computation methods for obtaining non-trivial common subsumers discussed in the last chapter, we have supplied the theoretical groundwork to support the bottom-up extension and the customization of knowledge bases in the presence of disjunction. To put these approaches to practice and to assess their usefulness and performance and to compare them, we need to implement them in a non-naive way. This implementation is described in the next chapter.

# Chapter 6

# Implementations of non-standard inferences

The methods for the computation of common subsumers developed in the last chapters are designed to be employed in practice. To this end it is of course necessary to implement and evaluate them. Many of the inferences proposed in this work are investigated and implemented for the first time. They need to be evaluated with respect to their usefulness for the intended application. Moreover, in order to be useful in the application of extending DL knowledge bases, where direct user interaction is required, the implementation of the methods proposed here should be efficient. In this chapter we describe our implementation of the non-standard inferences employed for the computation of non-trivial common subsumers in the presence of disjunction. Our implementation aims at demonstrating the feasibility of the approaches proposed here. It is also an efficient implementation in terms of the algorithms used. It is not focussed on 'low-level' optimizations, such as particular data-structures or sophisticated encodings of concept descriptions.

We divide the inferences into two groups based on the similarity of their computation algorithms:

**Generalization inferences**, which comprise the lcs, approximation, acs and scs.[23] The computation algorithms for these inferences are all cross-product based and mainly differ in the way they treat concept names and in the way they prepare concept description for recursive calls.

**Syntactic inferences**, which comprise the (heuristics for the) difference operator and minimal rewriting.

We concentrate in this chapter on the generalization algorithms, since they are crucial for the computation of common subsumers. We first investigate optimization techniques for this group of algorithms in Section 6.2 and then describe their (possible) implementation in Section 6.3. We also describe the implementation of the syntactic

[23]The implementation of the gcs as defined in Section 5.4.1 is discussed in the forthcoming thesis of Sertkaya [Ser07] and is omitted in this thesis.

inferences briefly. All of the described implementations are part of the non-standard inference system Sonic. This system enables the use of non-standard inferences discussed in this thesis via an ontology editor plug-in and implements a prototype version for a user interface of the knowledge base extension approaches bottom-up construction, customization of background knowledge bases as well as extension by modification.

## 6.1 Implementations of DL systems

The implementation of the first DL reasoning systems were based on structural methods for computing inferences as satisfiability or subsumption w.r.t. unfoldable TBoxes. From the mid-nineties on, most DL systems were implemented as tableaux-based systems. Here, in order to disprove a subsumption relationship $C \sqsubseteq D$ the reasoner tries to construct a counter-model that serves as a witness for the satisfiability of $C \sqcap \neg D$. During the construction of the counter-model, the reasoner has to explore a search space determined by the alternatives in the concept description induced by, for example, disjunction. The tableaux-based reasoning method has been implemented in highly optimized ways in most of recent DL systems such as FaCT++ [TH06], RacerPro [Rac05] or Pellet [SP04] for very expressive DLs. These systems employ a number of optimization techniques that are either well-known from automated reasoning—such as dependency directed backtracking [Bak95] or semantic branching [DLL62; Fre95]—or techniques tailored to DL systems such as absorption of general TBox statements [HT00], lazy unfolding [BHN+92], model merging [Hor97; HMT01] or specific caching techniques [THPS07]. *Semantic branching* and *dependency directed backtracking* are techniques that avoid that parts of the search space which are known not to contain the solution, are explored repeatedly. When *lazy unfolding* is employed, the concept is not unfolded w.r.t. the TBox completely at the beginning of the computation, but is only unfolded for the top most role level, if examination of the definition of the concept is necessary. On the one hand this method saves memory, since complete unfolding of concepts can be avoided in many cases. On the other hand, inconsistencies can be detected faster if a concept name and its negation appear in a concept description directly. *GCI absorption* transforms a certain sub-class of general concept inclusion statements from the TBox into a form that can be treated as concept definitions during reasoning. For reasoning with GCIs in general it is necessary to introduce more disjunctions, which in turn increase the search space to be explored by the tableau method drastically. So, by GCI absorption this blow-up of the search space can be limited. *Model merging* and *caching* re-use obtained results in a smart way. Caching reuses the results obtained from previous satisfiability tests directly. Model merging is a structural method that is employed to test satisfiability by means of a model generated in previous tableaux tests. This method is sound, but not complete since only one model is considered among many possible ones. In case the model merging cannot prove that $C \sqcap \neg D$ holds, the tableau method is employed.

Todays DL systems are implemented in the programming languages Common Lisp,

Java or C++. All of these implementations use special data structures to encode concepts and concept descriptions in a compact way such that large knowledge bases can be stored and accessed efficiently. In particular, by the use of these data structures known subsumption relationships can be retrieved fast when checking for new subsumption relationships or when classifying the whole TBox.

The implementations of standard reasoning tasks based on the aforementioned optimization techniques have shown a drastic gain in performance despite the computational complexity of the reasoning methods [HPS99; HM01a; THPS07]. However, for non-standard inferences it is less probable to achieve such a gain for two reasons. First, non-standard inferences are computation problems, while (most) standard inferences are decision problems. Simply by the fact that many of the methods for computing common subsumers can generate output concept descriptions exponential in the size of the input, it is clear that optimization cannot avoid exponential runtimes in principle.[24] The second reason why achieving a drastic gain in performance comparable to the one achieved for standard reasoning is rather not probable is that most methods for computing NSIs are purely constructive and do not have to consider a search space opened up by alternatives to explore and thus optimizations like Backjumping or Semantic Branching cannot be employed. Nevertheless, the computation algorithms devised in the last three chapters of this thesis for the computation of non-trivial common subsumers definitely leave room for optimizations.

## 6.2 Optimization techniques for generalization inferences

In this section we devise optimization techniques for the computation algorithms for the generalization algorithms introduced in the last chapters. The group of generalization algorithms comprises the lcs, approximation, acs and scs. The optimization techniques we propose are mainly motivated by our application scenario, where these inferences are used in an interactive way. Thus our optimizations aim at shorter run-times. We do not focus on a better use of storage space.

### 6.2.1 Lazy unfolding and lazy normalization

If a generalization inference is applied to concept descriptions containing concept names defined with respect to an unfoldable TBox, the concept description is unfolded, i.e., names of defined concepts are replaced by their definition from the TBox. Furthermore, concept descriptions are transformed into normal form in a preprocessing step. These two preprocessing steps are costly, since each of them can result in concept descriptions exponential in the size of the initial concept description. However, these two steps are not always necessary for all sub-concept descriptions an input concept description contains, since these sub-concept descriptions might not contribute to the overall result.

[24] Nevertheless, most of the examples for the worst case complexities for the computation of common subsumers exhibit a very regular structure (e.g. see Example 20 on page 45 or Example 27 on page 55). It seems very unlikely that these cases are encountered when applying the computation of common subsumers to concepts from knowledge bases written by human users.

For standard reasoning tasks [BFH+94; Hor97] and also for the computation of the lcs [BT02a] the first source of complexity can often be alleviated by *lazy unfolding*. The idea is to replace a defined concept in a concept description only if examination of that part of the description is necessary during computation. *Lazy unfolding* unfolds all defined concepts appearing on the top-level of the concept description under consideration while defined concepts on deeper role levels remain unchanged as long as possible. We refer to the complete unfolding procedure as *eager unfolding*.

When computing the lcs, the main benefit of lazy unfolding is that in some cases defined concepts can be used directly in the lcs concept description. If, for example, a defined concept name $C$ appears in all input concept descriptions on the same role-level, the concept definition of $C$ does not need to be processed, but the name $C$ can be inserted into the lcs directly, see [BT02a] for details. The experiments carried out in [BT02a] indicate that the computation of the lcs takes one third of the time when lazy unfolding is employed in comparison to eager unfolding. The lazy unfolding procedure can be used as well for $\mathcal{ALE}$-unfolding during the computation of the scs.

In the case of concept approximation, however, this use of defined names cannot be utilized even if a defined concept is obviously common to all disjuncts. For example, in $(A \sqcap C) \sqcup (C \sqcap \neg B)$ the concept name $C$ cannot be used directly as a name in the concept approximation, because the $\mathcal{ALC}$-concept description that $C$ stands for must be approximated first. Thus eager unfolding can only be avoided for approximation if one of the disjuncts has a deeper maximal role level than the other disjuncts. In this case the concept names on these role levels need not be taken into account when computing the approximation; thus they need not be unfolded.

Similarly to unfolding, normalization can be performed 'on demand', which obviously saves computation time if a sub-concept description does not need to be considered during further computation. In particular for the computation of approximation and for the acs it is desirable to perform lazy normalization, since the normal forms for $\mathcal{ALC}$ and $\mathcal{ALCN}$ can result in concept descriptions exponential in the size of the initial one. Like for the normalization of DLs offering number restrictions, lazy normalization promises a substantial gain in computation time, since the normalization here requires lcs calls to obtain the induced existential restrictions and the induced value restrictions. In particular this is makes a difference for $\mathcal{ALEN}$, since here the lcs cannot be obtained by simply using disjunction, but by employing the $\mathcal{ALEN}$-lcs computation—a method that might return concept descriptions of size double exponential in the size of the input.

The implementation of lazy unfolding and normalization for the computation of computing generalization inferences simply requires that at the beginning of the computation (and consequently at the beginning of each recursive call) the concept description is unfolded on the top most role level and normalized for this role level. Thus the preprocessing steps of the original procedure are executed at the beginning of each (recursive) call.

By implementing lazy unfolding and lazy normalization we cannot only save computation time of unnecessary preprocessing, but, in addition, the concept descriptions that need to be used and stored during computation are considerably smaller, which saves storage space. In contrast to other optimization techniques, lazy unfolding does

not generate any overhead. Thus by the use of lazy unfolding and lazy normalization the generalization inferences can be implemented in a way that is more efficient at no extra costs.

### 6.2.2 Conjunct-wise computation for nice concepts

The double-exponential time complexity of the approximation algorithm suggests the following approach to optimization: in comparison to approximating an input concept description $C$ as a whole, a significant amount of time could be saved by splitting $C$ into its conjuncts and approximating them separately. If, for instance, $C$ consists of two conjuncts of size $n$ then the approximation of $C$ takes some $a^{b^{2n}}$ steps while the conjunct-wise approach would just take $2a^{b^n}$. Unfortunately, splitting an arbitrary input concept at conjunctions leads to incorrect approximations. For example, the approximation $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(\exists r.\top \sqcap (\forall r.A \sqcup \exists r.A))$ yields $\exists r.A$ while the conjunct-wise version $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(\exists r.\top) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALE}}(\forall r.A \sqcup \exists r.A)$ only produces $\exists r.\top$. In general, the computation of an approximation cannot be split at the conjunction because of possible interactions—in case of $\mathcal{ALC}$-$\mathcal{ALE}$-approximation between existential and value restrictions on the one hand and inconsistencies induced by negation on the other. In [BT02b] those concept descriptions were called *nice* for which this splitting strategy still produces the correct result.

**Definition 73 (Nice concepts).** Let $C := C_1 \sqcap \cdots \sqcap C_n$ be a $\mathcal{L}_1$-concept description. $C$ is *nice* if $\mathsf{approx}_{\mathcal{L}_2}(C) \equiv \mathsf{approx}_{\mathcal{L}_2}(C_1) \sqcap \cdots \sqcap \mathsf{approx}_{\mathcal{L}_2}(C_n)$. ◇

For these concept descriptions interactions between conjuncts are excluded. Since the use of nice concept descriptions is to speed-up approximation, it is important that the conditions for distinguishing these concept descriptions can be tested easily. Therefore the test for nice concept descriptions should be based on simple discrimination conditions.

In the following we extend an approach introduced by us in [BT02b], where sound, but not complete conditions were given to detect concept descriptions from the class of nice $\mathcal{ALC}$-concept descriptions. For this class of concepts the conjunct-wise approach to approximation and, in a similar form, to the lcs produces the correct result. In this section we first discuss the syntactic conditions for nice $\mathcal{ALC}$-concepts from [BT02b] and, second, relax and extend these conditions to $\mathcal{ALCN}$-concepts.

#### Conditions for nice $\mathcal{ALC}$-concept descriptions

Nice $\mathcal{ALC}$-concept descriptions can be detected by the following sufficient conditions for $\mathcal{ALC}$-concept descriptions:

1. the existential or value restrictions are limited to one type per role-depth: on every role depth of a nice concept either no $\forall$-restrictions or no $\exists$-restrictions occur.

2. a concept name and its negation may not occur on the same role-depth of a nice concept.

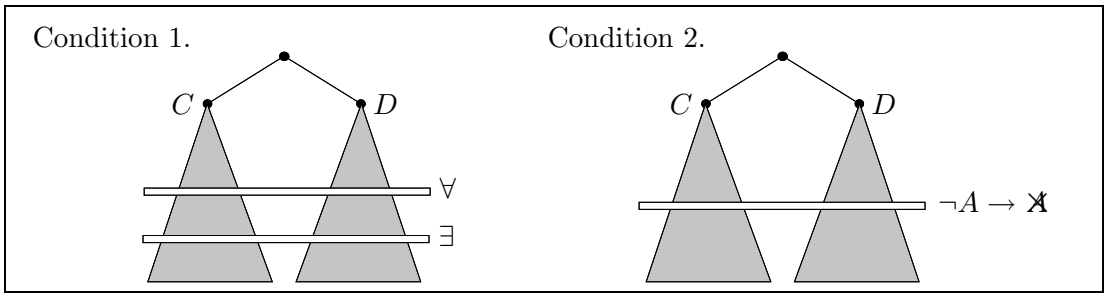


Figure 6.1: Strict conditions for nice concepts

Consider Figure 6.1 for an illustration of these rules. In [BT02b] these conditions are formally defined and it is shown that for $\mathcal{ALC}$-concept descriptions fulfilling these conditions, conjunct-wise approximation is correct. The above conditions are intuitive and easy to test, but very strict. Consider the concept description $(\exists r.\exists s.A \sqcup B) \sqcap (\exists t.\forall s.\neg A \sqcup C)$, which violates both conditions. However, we obtain the correct result if the conjuncts are approximated independently, since the relevant concept descriptions are nested in existential restrictions for different roles. In general, a concept description can still be approximated conjunct-wise, if the 'interacting' concept descriptions are reachable via different role paths. Too strict conditions to distinguish nice concept descriptions would rule out too many concept descriptions that could actually be approximated in a conjunct-wise way. In these cases the 'expensive' approximation must be applied. Next, in order to be able to detect more concept descriptions as nice, we devise relaxed conditions for nice concept descriptions that also can handle number restrictions.

### Relaxed conditions for nice $\mathcal{ALCN}$-concept descriptions

The conditions for nice $\mathcal{ALCN}$-concept descriptions have to take into account the information induced by number restrictions in combination with other concept constructors. As we saw in Section 3.3, we can obtain induced existential restrictions by at-least restrictions. Induced value restrictions can be obtained by at-most restrictions (in combination with existential restrictions). Furthermore, at-most restrictions and at-least restrictions can imply contradictions, which in turn can lead to disjuncts equivalent to $\bot$ and thus deterministic disjunctions. To get a better understanding of the possible interactions consider the following example.

**Example 74.** *Let $C = C_1 \sqcap C_2$ be an $\mathcal{ALCN}$-concept description, where the conjuncts are defined as:*

$$\begin{aligned} C_1 &= (\forall r.\bot) \sqcap (\exists s.A) \sqcap (\exists s.\neg A) \sqcup \bot, \\ C_2 &= (\geq 1\ r) \sqcup (\leq 1\ s) \sqcup B. \end{aligned}$$

*Now, if we approximate the concept $C$ conjunct-wise we obtain* $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_1) = (\forall r.\bot) \sqcap (\exists s.A) \sqcap (\exists s.\neg A)$ *and* $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_2) = \top$*, yielding $(\forall r.\bot) \sqcap (\exists s.A) \sqcap (\exists s.\neg A) \sqcap \top$ as the combined result. Due to the incompatibilities between $C_1$ and $C_2$*

*for the information on the roles $r$ and $s$, the disjunction in $C_2$ collapses to $B$. The approximation yields* $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) = (\forall r.\bot) \sqcap (\exists s.A) \sqcap (\exists s.\neg A) \sqcap B$*, which is more specific.*

The example shows that the conditions for nice $\mathcal{ALC}$-concept descriptions do not suffice to detect nice $\mathcal{ALCN}$-concept descriptions. We need to take into account the number restrictions appearing on the top-level of $C$. We introduce a notation to access the numbers within number restrictions. For an $\mathcal{ALCN}$-concept description $C$ and a role $r$ let

$\mathsf{at\text{-}least}_r(C)$ denote the maximal number appearing in all at-least restrictions on the top-level of $C$ or 0, if no at-least restriction appears on the top-level of $C$ and

$\mathsf{at\text{-}most}_r(C)$ denote the minimal number appearing in all at-most restrictions on the top-level of $C$ or $\infty$, if no at-most restriction appears on the top-level of $C$ .

Next, we specify the notion of sub-concept descriptions accessible by a role path, which will be used in the conditions for nice concept descriptions.

**Definition 75.** Let a *$Qr$-path* (denoted $\rho$) be defined as $\rho = [Q\, r]^*$ for $Q \in \{\exists, \forall\}$ and $r \in N_R$ and let $\lambda$ denote the empty $Qr$-path.

Let $C$ be an $\mathcal{ALC}$-concept description and $\rho$ be a $Qr$-path, then we define the set of *$\rho$-reachable sub-concepts* of $C$ as:

$$sub(C, \rho) := \begin{cases} prim(C) & \text{if } \rho = \lambda, \\ sub(\mathsf{val}_r(C), \rho') & \text{if } \rho = \forall\, r \cdot \rho', \\ \bigcup_{C' \in \mathsf{ex}_r(C)} sub(C', \rho') & \text{if } \rho = \exists\, r \cdot \rho', \\ \emptyset & \text{otherwise.} \end{cases}$$

◇

Formally, we can detect nice $\mathcal{ALCN}$-concept descriptions by the conditions given in the following definition:

**Definition 76 (Sufficient conditions for nice $\mathcal{ALCN}$-concept descriptions).** Let $C$ be an $\mathcal{ALCN}$-concept description in NNF. Then $C$ is *nice*, if for every $Qr$-path $\rho$ with $C_1, C_2 \in sub(C, \rho)$ and $C_1', C_2'$ denoting $C_1, C_2$ in $\mathcal{ALCN}$-normal form and all $r \in N_R$ it holds that

1. $|\{\exists \mid \mathsf{at\text{-}least}_r(C) \neq 0\} \cup \{\exists \mid \mathsf{ex}_r(C_1) \cup \mathsf{ex}_r(C_2) \neq \emptyset\}| +$
   $|\{\forall \mid \mathsf{at\text{-}most}_r(C) \neq \infty\} \cup \{\forall \mid \bigsqcap_{i \in \{1,2\}} \mathsf{val}_r(C_i) \not\equiv \top\}| \leq 1$ and
2. $prim(C_1) \cup prim(C_2)$ does not contain a concept name and its negation.

◇

Intuitively, Condition 1 guarantees for all sub-concept descriptions reachable via the same $Qr$-path that either only restrictions are used that require role successors, i.e., at least or existential restrictions, or that only restrictions are used that constrain all role successors of the same role, i.e., at-most or value restrictions. A visualization of

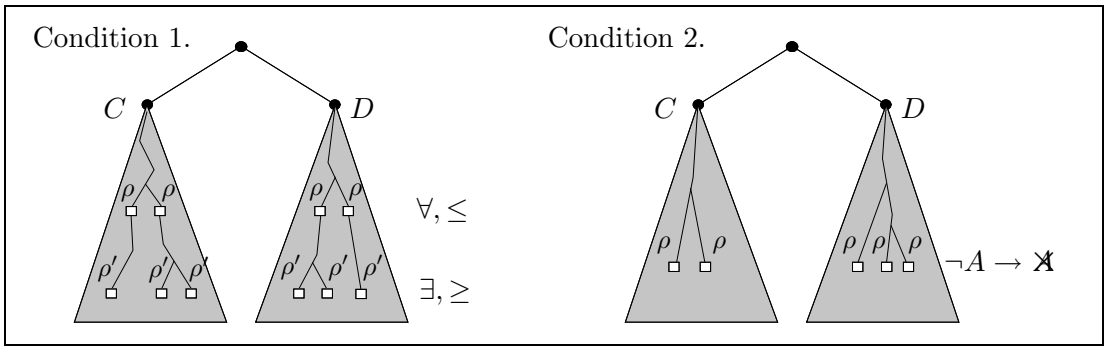


Figure 6.2: Conditions for nice $\mathcal{ALCN}$-concept descriptions.

the conditions from Definition 76 is depicted in Figure 6.2. It remains to be shown that nice concepts as defined above in fact have the desired property. In preparation for this we firstly present a simple set-theoretic result which later on will allow us to reduce the number of existential restrictions computed in an approximation of nice concepts.

The distribution of a conjunction over a disjunction in the $\mathcal{ALCN}$-normalization produces conjunctions of a very regular structure. As an example, consider the concept $E := (C_1 \sqcup C_2) \sqcap (D_1 \sqcup D_2)$ with $C_i := \exists r.C_i'$ and $D_j := \exists r.D_j'$. Assuming that all existential restrictions are $\mathcal{ALEN}$-concepts, the normalization returns $\bigsqcup_{i,j}(C_i \sqcap D_j)$. The approximation algorithm then computes the lcs over every combination of existential restrictions from the four disjuncts. Nevertheless, every existential restriction in the result $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(E)$ either subsumes $\exists r.\mathsf{lcs}(C_1', C_2')$ or $\exists r.\mathsf{lcs}(D_1', D_2')$ because it corresponds to the lcs of a superset of one of the above sets. The following lemma shows that this subset-superset property can be generalized, the proof was given in [BT02c].

**Lemma 77.** *Let $m, n \in \mathbb{N}$. For every $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$, let $A_i$ and $B_j$ be arbitrary finite sets, let $U_{ij} := A_i \cup B_j$, and let $u_{ij} \in U_{ij}$. Denote by $U$ the set of all $u_{ij}$, i.e., $U := \{u_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. Then one of the following claims holds: either, for every $i$ there exist elements $a_i \in A_i$ with $\{a_i \mid 1 \leq i \leq m\} \subseteq U$; or, for every $j$ there exist $b_j \in B_j$ with $\{b_j \mid 1 \leq j \leq n\} \subseteq U$.*

The choice of sets in the unions $U_{ij}$ in the above lemma corresponds to tuples in the product $\{A_1, \ldots, A_m\} \times \{B_1, \ldots, B_n\}$. The claim can be generalized to $n$-ary products where every union corresponds to a tuple from $\{S_{11}, \ldots, S_{1k_1}\} \times \cdots \times \{S_{n1}, \ldots, S_{nk_n}\}$. The following lemma provides the more general result.

**Lemma 78.** *For every $1 \leq i \leq n$ and $1 \leq j_i \leq k_i$, let $S_{ij_i}$ be an arbitrary set. For every tuple $\bar{t}$ in the set $T := \{1, \ldots, k_1\} \times \cdots \times \{1, \ldots, k_n\}$, denote by $U_{\bar{t}}$ the union $\bigcup_{i=1}^{n} S_{i\bar{t}(i)}$ (with $\bar{t}(i)$ denoting the $i$th component of $\bar{t}$). For every $\bar{t} \in T$, let $u_{\bar{t}} \in U_{\bar{t}}$. Let $U := \{u_{\bar{t}} \mid \bar{t} \in T\}$. Then there exists an index $i \in \{1, \ldots, n\}$ and elements $s_{ij} \in S_{ij}$ for $1 \leq j \leq k_i$ such that the set $\{s_{ij} \mid 1 \leq j \leq k_i\}$ is a subset of $U$.*

Again, the proof can be found in [BT02c]. By means of the above lemma we can show that the lcs of sets of $\mathcal{ALEN}$-concepts obeying the conditions from Definition 76

of a certain form can be simplified. The following example motivates the relevant case. The $\mathcal{ALCN}$-normal form of a nice concept of the form $(C_1 \sqcup \cdots \sqcup C_m) \sqcap (D_1 \sqcup \cdots \sqcup D_n)$ with no further disjunction on the top-level of all $C_i$ and $D_j$ results in a disjunction of the form $\bigsqcup_{i,j} C_i \sqcap D_j$. Assume that all sub-concepts $C_i$, $D_j$ have only existential restrictions on top-level. For the approximation of this concept, computing the resulting existential restrictions requires to compute the lcs of (the approximation of) every combination of existential restrictions from the relevant disjuncts. Thus, for every pair $(i, j)$, every existential restriction $E_{ij} \in \mathsf{ex}_r(C_i \sqcap D_j)$ is approximated, then the lcs over all $\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(E_{ij}) \mid i, j\}$ is computed. Since $\mathsf{ex}_r(C_i \sqcap D_j)$ equals the union $\mathsf{ex}_r(C_i) \cup \mathsf{ex}_r(D_j)$, the previous lemma can be employed to restrict the lcs to a much smaller set. The following lemma provides the exact proof.

**Lemma 79.** *For $1 \leq i \leq 2$, let $C_i$ and $D_i$ be $\mathcal{ALEN}$-concept descriptions such that $C_1 \sqcap C_2 \sqcap D_1 \sqcap D_2$ is a nice concept description. Then it holds that $\mathsf{lcs}(\{C_i \sqcap D_j \mid i, j \in \{1, 2\}\}) \equiv \mathsf{lcs}(\{C_1, C_2\}) \sqcap \mathsf{lcs}(\{D_1, D_2\})$.*

**Proof.** We assume w.l.o.g. that $N_R = \{r\}$. We proceed with a proof by induction over the maximum role-depth $d$ of all $C_i$, $D_j$.

*Base case $d = 0$:* We have either $E_k = \bigsqcap_{A \in prim(E_k)} A \sqcap \bigsqcap_{r \in N_R} (\leq \mathsf{at\text{-}most}_r(E_k)\ r)$ or we have $E_k = \bigsqcap_{A\ \in\ prim(E_k)} A \sqcap \bigsqcap_{r \in N_R} (\geq \mathsf{at\text{-}least}_r(E_k)\ r)$ for $E_k \in \{C_1, C_2, D_1, D_2\}$. W.l.o.g. we assume the first case, the second is analogous. The definition of nice guarantees that no inconsistencies can be introduced by a combination of an atomic concept and its negation. Hence, the $\mathsf{lcs}(\{C_i \sqcap D_j \mid i, j \in \{1, 2\}\})$ then yields $\bigsqcap_{A \in S} A \sqcap (\leq m\ r)$, where

$$m = \max\{\mathsf{max}_r(C_i \sqcap D_j) \mid i, j \in \{1, 2\}\} = \max\{\min\{\mathsf{at\text{-}most}_r(C_i), \mathsf{at\text{-}most}_r(D_j)\} \mid i, j \in \{1, 2\}\}$$

and where $S$ is the intersection of all sets of primitive concepts of the form $prim(C_i \sqcap D_j)$. Hence, $S$ equals $\bigcap\{prim(C_i) \cup prim(D_j) \mid i, j \in \{1, 2\}\}$. By distributing the intersection over the union, $S$ can be expressed as the union $(prim(C_1) \cap prim(C_2)) \cup (prim(D_1) \cap prim(D_2))$. The conjunction $\bigsqcap_{A \in S} A$ is therefore equivalent to the conjunction $\bigsqcap_{A \in prim(C_1) \cap prim(C_2)} A \sqcap \bigsqcap_{A \in prim(D_1) \cap prim(D_2)} A$. Due to the distributivity of the maximum operation over the minimum operation, we obtain for the number in the at-most restrictions:

$$\max\big\{\min\{\mathsf{at\text{-}most}_r(C_i), \mathsf{at\text{-}most}_r(D_j)\} \mid i, j \in \{1, 2\}\big\} = \min\big\{\max\{\mathsf{at\text{-}most}_r(C_1), \mathsf{at\text{-}most}_r(C_2)\},\ \max\{\mathsf{at\text{-}most}_r(D_1), \mathsf{at\text{-}most}_r(D_2)\}\big\}.$$

Thus, we obtain for the whole conjunction of (negated) concept names and at-most restrictions:

$$\bigsqcap_{A \in prim(C_1) \cap prim(C_2)} A \sqcap (\geq\ \max\{\mathsf{at\text{-}most}_r(C_1), \mathsf{at\text{-}most}_r(C_2)\}\ r)$$

$$\bigsqcap_{A \in prim(D_1) \cap prim(D_2)} A \sqcap (\geq\ \max\{\mathsf{at\text{-}most}_r(D_1), \mathsf{at\text{-}most}_r(D_2)\}\ r)$$

By definition of the lcs, this conjunction is equivalent to the conjunction of $\mathsf{lcs}(\{C_i \mid 1 \leq i \leq 2\})$ and $\mathsf{lcs}(\{D_j \mid 1 \leq j \leq 2\})$.

*Induction step $d > 0$:* Since $C_1 \sqcap \cdots \sqcap D_2$ is nice, two cases are distinguished, depending on the restrictions on roles and number of role successors appearing on the outermost role level. In the first case, all $C_i$ and $D_i$ are of the form $\bigsqcap_{A \in prim(C_i)} A \sqcap \forall r.C_i' \sqcap (\leq \mathsf{at\text{-}most}_r(C_i)\ r)$ and $\bigsqcap_{A \in prim(D_j)} A \sqcap \forall r.D_j' \sqcap (\leq \mathsf{at\text{-}most}_r(D_j)\ r)$, respectively. Then, $\mathsf{lcs}(\{C_i \sqcap D_j \mid i,j \in \{1,2\}\})$ is defined as

$$\bigsqcap_{A \in S} A \ \sqcap\ \forall r.\mathsf{lcs}(\{C_i' \sqcap D_j' \mid i,j \in \{1,2\}\}) \ \sqcap\ (\leq m\ r),$$

where $S$ again equals $\bigcap\{prim(C_i) \cup prim(D_j) \mid i,j \in \{1,2\}\}$. Analogously to the case of $d = 0$, the set $S$ can be expressed as the union of $prim(C_1) \cap prim(C_2)$ and $prim(D_1) \cap prim(D_2)$. For the at-most restriction we have $m = \max\{\mathsf{max}_r(C_i \sqcap D_j) \mid i,j \in \{1,2\}\} = \max\{\min\{\mathsf{at\text{-}most}_r(C_i), \mathsf{at\text{-}most}_r(D_j)\} \mid i,j \in \{1,2\}\}$ we obtain for the numbers in the at-most restrictions:

$$\begin{aligned}&\max\big\{\min\{\mathsf{at\text{-}most}_r(C_i), \mathsf{at\text{-}most}_r(D_j)\} \mid i,j \in \{1,2\}\big\} = \\ &\min\big\{\max\{\mathsf{at\text{-}most}_r(C_1), \mathsf{at\text{-}most}_r(C_2)\},\ \max\{\mathsf{at\text{-}most}_r(D_1), \mathsf{at\text{-}most}_r(D_2)\}\big\}.\end{aligned}$$

Since $C_1 \sqcap C_2 \sqcap D_1 \sqcap D_2$ is a nice concept description containing value restrictions, we have no existential or at-least restrictions in the top-level of $C_1 \sqcap C_2 \sqcap D_1 \sqcap D_2$. Thus $\mathsf{ind\text{-}val}_r(E_i) = \mathsf{val}_r(E_i)$, if $\mathsf{at\text{-}most}_r(E_i) = 0$ for $E_i \in \{C_1, C_2, D_1, D_2\}$. Due to the induction hypothesis, the lcs in the value restriction is equivalent to $\mathsf{lcs}(\{C_i \mid 1 \leq i \leq 2\}) \ \sqcap\ \mathsf{lcs}(\{D_j \mid 1 \leq j \leq 2\})$. Since a conjunction in a value restriction may be split into a conjunction of value restrictions, we obtain for the entire conjunction:

$$\begin{aligned}&\bigsqcap_{A \in prim(C_1) \cap prim(C_2)} A \sqcap \bigsqcap_{A \in prim(D_1) \cap prim(D_2)} A \sqcap \\ &\forall r.\mathsf{lcs}(\{C_i' \mid 1 \leq i \leq 2\}) \ \sqcap\ \forall r.\mathsf{lcs}(\{D_j' \mid 1 \leq j \leq 2\}) \sqcap \\ &(\leq\ \max\{\mathsf{at\text{-}most}_r(C_1), \mathsf{at\text{-}most}_r(C_2)\}\ r) \sqcap \\ &(\leq\ \max\{\mathsf{at\text{-}most}_r(D_1), \mathsf{at\text{-}most}_r(D_2)\}\ r).\end{aligned}$$

According to the definition of the lcs, this expression can be written as

$$\mathsf{lcs}(\{C_i \mid 1 \leq i \leq 2\}) \sqcap \mathsf{lcs}(\{D_j \mid 1 \leq j \leq 2\}).$$

In the second case, all $\mathcal{ALCN}$-concept descriptions $C_i$ and $D_j$ are concept descriptions of the form $\bigsqcap_{A \in prim(C_i)} A \sqcap \bigsqcap_{C_i' \in \mathsf{ex}_r(C_i)} \exists r.C_i' \sqcap (\geq \mathsf{at\text{-}least}_r(C_i)\ r)$ and $\bigsqcap_{A \in prim(D_j)} A \sqcap \bigsqcap_{D_j' \in \mathsf{ex}_r(D_j)} \exists r.D_j' \sqcap (\geq \mathsf{at\text{-}least}_r(D_j)\ r)$, respectively. Since $C$ is nice and contains existential or at-least restrictions, it does not contain at-most restrictions, thus the only existential mapping induced is the identity mapping. Thus $\mathsf{ind\text{-}ex}_r(E_i) = \mathsf{ex}_r(E_i)$ for all $E_i \in \{C_1, C_2, D_1, D_2\}$. The least common subsumer $\mathsf{lcs}(\{C_i \sqcap D_j \mid i,j \in \{1,2\}\})$

then yields

$$\prod_{A \in S} A \ \sqcap \prod_{\substack{E_1 \in \mathsf{ex}_r(C_1) \cup \mathsf{ex}_r(D_1),\\ E_2 \in \mathsf{ex}_r(C_1) \cup \mathsf{ex}_r(D_2),\\ E_3 \in \mathsf{ex}_r(C_2) \cup \mathsf{ex}_r(D_1),\\ E_4 \in \mathsf{ex}_r(C_2) \cup \mathsf{ex}_r(D_2)}} \mathsf{lcs}(\{E_i \mid 1 \leq i \leq 4\}) \ \sqcap \ (\geq m\ r) \tag{6.1}$$

with $S$ as before. The number $m$ is obtained in the dual way to the previous case $m = \min\{\mathsf{min}_r(C_i \sqcap D_j) \mid i,j \in \{1,2\}\} = \min\{\max\{\mathsf{at\text{-}least}_r(C_i), \mathsf{at\text{-}least}_r(D_j)\} \mid i,j \in \{1,2\}\}$. Due to distributivity, we obtain:

$$\begin{aligned}&\min\big\{\max\{\mathsf{at\text{-}least}_r(C_i), \mathsf{at\text{-}least}_r(D_j)\} \mid i,j \in \{1,2\}\big\} = \\ &\max\big\{\min\{\mathsf{at\text{-}least}_r(C_1), \mathsf{at\text{-}least}_r(C_2)\},\ \min\{\mathsf{at\text{-}least}_r(D_1), \mathsf{at\text{-}least}_r(D_2)\}\big\}.\end{aligned}$$

It is shown in Lemma 78 that every set $\{E_i \mid 1 \leq i \leq 4\}$ in the conjunction (6.1) is a superset of a set either of the form $\{C_1', C_2'\}$ with all $C_i' \in \mathsf{ex}_r(C_i)$ or of the form $\{D_1', D_2'\}$ with all $D_j' \in \mathsf{ex}_r(D_j)$. Due to the monotonicity of the lcs (w.r.t. subsumption) every expression $\mathsf{lcs}(\{E_i \mid 1 \leq i \leq 4\})$ is therefore more general than either $\mathsf{lcs}(\{C_1', C_2'\})$ or $\mathsf{lcs}(\{D_1', D_2'\})$ for appropriate existential restrictions $C_1', C_2'$ or $D_1', D_2'$. Conversely, every set of the form $\{C_1', C_2'\}$ and $\{D_1', D_2'\}$ occurs in the above conjunction as one choice of $E_i$, with $1 \leq i \leq 4$. Hence, the above existential restrictions can be simplified, yielding

$$\prod_{\substack{E_1 \in \mathsf{ex}_r(C_1),\\ E_2 \in \mathsf{ex}_r(C_2)}} \mathsf{lcs}(E_1, E_2) \ \sqcap \prod_{\substack{E_3 \in \mathsf{ex}_r(D_1),\\ E_4 \in \mathsf{ex}_r(D_2)}} \mathsf{lcs}(E_3, E_4).$$

Analogously to the previous case, the set $S$ of atomic concepts can be written as a union, yielding the following conjunction

$$\begin{aligned}&\prod_{A \in prim(C_1) \cap prim(C_2)} A \ \sqcap \prod_{A \in prim(D_1) \cap prim(D_2)} A \ \sqcap \\ &\prod_{\substack{E_1 \in \mathsf{ex}_r(C_1),\\ E_2 \in \mathsf{ex}_r(C_2)}} \mathsf{lcs}(E_1, E_2) \ \sqcap \prod_{\substack{E_3 \in \mathsf{ex}_r(D_1),\\ E_4 \in \mathsf{ex}_r(D_2)}} \mathsf{lcs}(E_3, E_4) \ \sqcap \\ &(\geq \ \min\{\mathsf{at\text{-}least}_r(C_1), \mathsf{at\text{-}least}_r(C_2)\}\ r) \ \sqcap \\ &(\geq \ \min\{\mathsf{at\text{-}least}_r(D_1), \mathsf{at\text{-}least}_r(D_2)\}\ r).\end{aligned}$$

By definition of the lcs, this is equivalent to

$$\mathsf{lcs}(\{C_i \mid 1 \leq i \leq 2\}) \ \sqcap \ \mathsf{lcs}(\{D_j \mid 1 \leq j \leq 2\}).$$

❑

The above claim can again be generalized to larger conjunctions. Observe that by Lemma 79 we in fact have a proof that the lcs can also be computed correctly in a conjunct-wise fashion for nice concept descriptions. Thus the method of conjunct-wise computation can also be employed to speed up the computation of the lcs.

We are now ready to show that approximating nice concept descriptions, as defined in Definition 76, can be simplified to a conjunction of approximations. For the sake of simplicity we restrict our attention to binary conjunctions.

**Theorem 80.** *Let $C \sqcap D$ be a nice $\mathcal{ALCN}$-concept description.*
*Then* $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D) \equiv \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D)$.

The claim is proved by induction over the sum of the nesting depths of $\sqcap$ and $\sqcup$ on every role level in $C$ and $D$. For the induction step, a case distinction is made depending on whether $C$ or $D$ are conjunctions or disjunctions. If at least one concept description is a disjunction the approximation is defined as the lcs of all $\mathcal{ALCN}$-normalized and approximated disjuncts (if one of the concepts is a conjunction, it firstly has to be distributed over the disjunction). The main idea then is to use Lemma 79 to transform single lcs calls of a certain form into a conjunction of lcs calls which eventually leads to the conjunction of the approximations of $C$ and $D$.

**Proof.** We give a proof by induction over the sum $n$ of the nesting depths of $\sqcap$ and $\sqcup$ on every role level in $C$ and $D$.

*Base case:* $n = 0$. No conjunction or disjunction occurs at any position in $C$ or $D$, implying that $C$ and $D$ are nice $\mathcal{ALEN}$-concept descriptions. Hence, $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \equiv C$ and $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D) \equiv D$. Since $C \sqcap D$ is also an $\mathcal{ALEN}$-concept description, we also know that $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D) \equiv C \sqcap D$. Consequently, $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D) \equiv \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D)$.

*Induction step:* $n > 0$. Due to the definition of nice, three cases have to be distinguished.

1. $C = \bigsqcap_{i=1}^{k} C_i$ and $D = \bigsqcap_{j=1}^{l} D_j$
   The approximation to be considered is $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}((\bigsqcap_{i=1}^{k} C_i) \sqcap (\bigsqcap_{j=1}^{l} D_j))$ which can be flattened. The nesting depth of the argument concept thus has decreased by 1 and we still have a nice concept. According to the induction hypothesis, the result is therefore equivalent to
   $$\bigsqcap_{i=1}^{k} \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_i) \sqcap \bigsqcap_{j=1}^{l} \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D_j)$$
   which in turn is equivalent to
   $$\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\bigsqcap_{i=1}^{k} C_i) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\bigsqcap_{j=1}^{l} D_j).$$
2. $C = \bigsqcup_{i=1}^{k} C_i$ and $D = \bigsqcup_{j=1}^{l} D_j$
   To compute $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D)$, the approximation algorithm at first transforms the input concept into $\mathcal{ALCN}$-normal form. The $\mathcal{ALCN}$-normal form of $C$

is of the form $\bigsqcup_{i'=1}^{k'} C'_{i'}$ with no disjunction on the top most role level of every $C'_{i'}$. Similarly, the $\mathcal{ALCN}$-normal form of $D$ yields an expression of the form $\bigsqcup_{j'=1}^{l'} D'_{j'}$. Hence, the $\mathcal{ALCN}$-normal form of $C \sqcap D$ is of the form

$$\bigsqcup_{1 \leq i' \leq k';\ 1 \leq j' \leq l'} (C'_{i'} \sqcap D'_{j'})$$

The approximation $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D)$ then by definition equals

$$\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'} \sqcap D'_{j'}) \mid 1 \leq i' \leq k', 1 \leq j' \leq l'\}).$$

As the maximum nesting depth in all of the occurring approximation expressions has decreased, we may exploit the induction hypothesis and obtain

$$\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'}) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D'_{j'}) \mid 1 \leq i' \leq k', 1 \leq j' \leq l'\}).$$

According to Lemma 79, the lcs can be split into two lcs-expressions of the form

$$\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'}) \mid 1 \leq i' \leq k'\}) \ \sqcap \ \mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D'_{j'}) \mid 1 \leq j' \leq l'\})$$

which by definition of the approximation are equivalent to the conjunction of two approximations, namely

$$\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\bigsqcup_{i'=1}^{k'} C'_{i'}) \ \sqcap \ \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\bigsqcup_{j'=1}^{l'} D'_{j'}),$$

which is equivalent to $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D)$, i.e., the separate approximation of the input concept descriptions.

3. $C = \bigsqcup_{i=1}^{k} C_i$ and $D = \prod_{j=1}^{l} D_j$
Similar to the previous case. The $\mathcal{ALCN}$-normal form of the input concept yields an expression of the form $\bigsqcup_{i'=1}^{k'}(C_i \sqcap \prod_{j'=1}^{l'} C'_{j'})$. The approximation $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C \sqcap D)$ therefore equals $\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'} \sqcap \prod_{j'=1}^{l'} D'_{j'})) \mid 1 \leq i' \leq k'\})$. According to the induction hypothesis the approximation can be split into $\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'}) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\prod_{j'=1}^{l'} D'_{j'}) \mid 1 \leq i' \leq k'\})$. Lemma 79 states that this lcs is equivalent to

$$\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C'_{i'}) \mid 1 \leq i' \leq k'\}) \ \sqcap \ \mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\prod_{j'=1}^{l'} D'_{j'}) \mid 1 \leq i' \leq k'\}).$$

The first lcs-expression is equivalent to the approximation of a disjunction while the second one contains $k'$ equal concepts. We thus obtain

$$\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\bigsqcup_{i'=1}^{k'} C'_{i'}) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\prod_{j'=1}^{l'} D'_{j'}).$$

Hence, we end up with the conjunction of the separate approximations of the form $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C) \sqcap \mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D)$.

❑

Due to Theorem 80 it is now possible to split the computation of approximations into independent parts. Although this does of course not change the complexity class of the approximation algorithm it is to be expected to be a significant benefit for actual implementations. The improved approximation algorithm is displayed in Figure 6.3. The algorithm requires the unfolded input concept to be in NNF. In the first step the $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ function checks whether the approximation is trivial. If it is not, the next step is to check whether the concept is nice. For nice concepts the $\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALEN}}$ function is invoked. For all other concepts the $\mathcal{ALCN}$-normal form is computed lazily, i.e., the conjunctions are distributed over the disjunctions only for the current top-level. Then the $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$ algorithm proceeds as before for concepts that are not nice. The $\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALEN}}$ function for nice concepts works similar. Having treated the trivial cases, the second step is to test if the concept is a conjunction. In that case the approximation is obtained by splitting the concept conjunct-wise and making a recursive call for each conjunct. For all other nice concepts the approximation is computed as in $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$, besides the recursive calls refer to $\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALEN}}$ and thus all embedded conjunctions are approximated in the conjunct-wise fashion.

Observe that the test conditions for nice concepts can be checked in linear time once the concept description is unfolded and in NNF. Unfolding and transforming the concept description into NNF always have to be performed to apply $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}$, i.e., testing whether a concept is nice is not much extra effort when approximating a concept.

If an $\mathcal{ALCN}$-TBox is to be translated into an $\mathcal{ALEN}$-TBox, the concept description on the right-hand side of each concept definition has to be replaced by its approximation. For practical applications with large TBoxes it is not feasible to perform such a translation in a naive way. The idea for optimizing this procedure is to re-use the approximation of a defined concept when approximating concept descriptions that in turn make use of this defined concept. More precisely, if we have already obtained the approximation of $C$ and want to compute the approximation of, e.g., $(D \sqcap \exists r.C)$, we would like to be able to insert the concept description $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C)$ directly into the right place in the concept description of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(D \sqcap \exists r.C)$. Unfortunately, this approach does not work for arbitrary $\mathcal{ALC}$-concept descriptions due to possible interactions between different parts of the concept description. Nice concepts, however, are defined in such way that rules out this kind of interaction. Hence, besides speeding up the computation of a single approximation, the property of being a nice concept also is a prerequisite for caching and more efficiently re-using already computed approximations. For example, if the defined concepts $C_1, C_2, C_3$ from the following TBox (with $A, B$ and $D$ as primitive concepts)

$$\begin{aligned}\mathcal{T} \quad = \{ \quad & C_1 = (\exists r.\neg A) \sqcup (\exists r.B), \\ & C_2 = \exists r.(\forall r.D \sqcup \neg E) \sqcap C_1 \sqcap \neg B, \\ & C_3 = \neg\,(\forall r.\exists r.(\neg D \sqcap A) \sqcup \neg C_1 \sqcup \neg C_2) \quad \}\end{aligned}$$

Input: unfolded $\mathcal{ALCN}$-concept description $C$ already in NNF
Output: upper $\mathcal{ALEN}$-approximation of $C$

c-approx$_{\mathcal{ALEN}}$

1. If $C \equiv \bot$, then c-approx$_{\mathcal{ALEN}}(C) := \bot$;
   if $C \equiv \top$, then c-approx$_{\mathcal{ALEN}}(C) := \top$
2. If nice-concept-p$(C)$ then return c-approx$_{\mathcal{ALEN}}(C) :=$ c-nice-approx$_{\mathcal{ALEN}}(C)$
3. Otherwise, transform the top-level of $C$ into $\mathcal{ALCN}$-normal form $C_1 \sqcup \cdots \sqcup C_n$ and return

   c-approx$_{\mathcal{ALEN}}(C) :=$

$$\prod_{A \in \bigcap_{i=1}^{n} prim(C_i)} A \ \sqcap$$

$$(\geq \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}\ r) \ \sqcap \ (\leq \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}\ r) \ \sqcap$$

$$\prod_{\substack{(C_1', \ldots, C_n') \in \\ \mathsf{ind\text{-}ex}_r(C_1) \times \cdots \times \mathsf{ind\text{-}ex}_r(C_n)}} \exists r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_i' \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}) \ \sqcap$$

$$\forall r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\mathsf{ind\text{-}val}_r(C_i)) \mid 1 \leq i \leq n\}).$$

c-nice-approx$_{\mathcal{ALEN}}$

1. If $C \equiv \bot$, then c-nice-approx$_{\mathcal{ALEN}}(C) := \bot$;
   if $C \equiv \top$, then c-nice-approx$_{\mathcal{ALEN}}(C) := \top$
2. If $C = C_1 \sqcap \cdots \sqcap C_n$, then return
   c-nice-approx$_{\mathcal{ALEN}}(C) := \prod_{i=1}^{n}$ c-nice-approx$_{\mathcal{ALEN}}(C_i)$
3. Otherwise, return

   c-nice-approx$_{\mathcal{ALEN}}(C) :=$

$$\prod_{A \in \bigcap_{i=1}^{n} prim(C_i)} A \ \sqcap$$

$$(\geq \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}\ r) \ \sqcap \ (\leq \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}\ r) \ \sqcap$$

$$\prod_{\substack{(C_1', \ldots, C_n') \in \\ \mathsf{ind\text{-}ex}_r(C_1) \times \cdots \times \mathsf{ind\text{-}ex}_r(C_n)}} \exists r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALEN}}(C_i' \sqcap \mathsf{val}_r(C_i)) \mid 1 \leq i \leq n\}) \ \sqcap$$

$$\forall r.\mathsf{lcs}_{\mathcal{ALEN}}(\{\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALEN}}(\mathsf{ind\text{-}val}_r(C_i)) \mid 1 \leq i \leq n\})$$

Figure 6.3: The improved algorithms c-approx$_{\mathcal{ALEN}}$ and c-nice-approx$_{\mathcal{ALEN}}$.

are to be approximated and $C_1$ is approximated first, then this concept description can be re-used in subsequent approximations. If unfolded and transformed into NNF the concepts $C_2$ and $C_3$ are nice concepts. Hence, the approximation of $C_2$ is the conjunction of $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(\exists r.(\forall r.D \sqcup \neg E))$ and $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(C_1)$ and $\mathsf{c\text{-}approx}_{\mathcal{ALEN}}(B)$, where the already computed approximation of $C_1$ can be inserted directly. For $C_3$ we can re-use both approximations of $C_1$ and $C_2$ directly and only have to compute the approximation of $\exists r.\forall r.(D \sqcup \neg A)$. Thus, the cost for approximating the entire TBox is reduced heavily.

#### Implementation of nice test

The implementation of approximation for nice $\mathcal{ALCN}$-concept descriptions requires few changes in the implementation of the approximation algorithm. The major part to implement is the function that tests for nice concept descriptions nice-concept-p. Since this test has to be performed at the beginning of every approximation computation, the implementation must be very efficient.

The procedure nice-concept-p in our implementation realizes the necessary conditions for nice $\mathcal{ALCN}$-concept descriptions from Definition 76. Our implementation of the nice test employs a couple of optimizations. Firstly, some steps are only taken on demand, such as unfolding and the transformation into NNF. Furthermore, nice-concept-p stores information of a certain named concept, say C, in a so-called info-table, where the key of this info-table is a path $\rho$ and the value is the sub(C,$\rho$). This enhances the checking of the conditions from Definition 76 'on-the-fly'. Suppose that while unfolding and transforming concept $C$, we encounter the concept name $A$ inspecting $Qr$-path $\rho$. Then, we update the info-table of $C$ by adding the concept name $A$ to the value of the key $\rho$. Before we add $A$, we first check whether adding $A$ violates Condition 2 in Definition 76. A similar procedure is carried out, if we encounter number, value and existential restrictions during the unfolding and transformation. Furthermore, our implementation does not only use dynamic programming to re-use results obtained during the current computation on whether a concept is already known to be nice or not, but caches these results. Based on this cache the already obtained information on whether a named concept is nice or not is re-used in subsequent runs of nice-concept-p.

### 6.2.3 Reduction of redundant recursive calls

All computation algorithms for generalization inferences invoke recursive calls for tuples from the cross product of the sets of (induced) existential restrictions of each input concept (or in each disjunct respectively). For instance, for the $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concept descriptions applied to the normalized concept description $C = C_1 \sqcup \ldots \sqcup C_n$ this set of recursive calls is determined by $(C'_1, \ldots, C'_n) \in \mathsf{ex}_r(C_1) \times \cdots \times \mathsf{ex}_r(C_n)$. In many cases it is likely that we obtain a set of redundant existential restrictions, if we compute the approximation for all elements of this set (after the propagation of value restrictions). This is especially likely when computing approximations, as the following example illustrates.

**Example 81 (Redundant recursive calls).** *Let $C = \exists r.B \sqcap \exists r.D \sqcap (A_1 \sqcup A_2 \sqcup A_3)$, then we obtain after the transformation into $\mathcal{ALC}$-normal form the concept description $C' = (A_1 \sqcap \exists r.B \sqcap \exists r.D) \sqcup (A_2 \sqcap \exists r.B \sqcap \exists r.D) \sqcup (A_3 \sqcap \exists r.B \sqcap \exists r.D)$. During the computation of the approximation of $C'$ a naive realization of the approximation algorithm would invoke calls for:*

$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B)) \tag{6.2}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D)) \tag{6.3}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B)) \tag{6.4}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D)) \tag{6.5}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B)) \tag{6.6}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D)) \tag{6.7}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(B)) \tag{6.8}$$
$$\mathsf{lcs}\ (\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D),\ \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D)) \tag{6.9}$$

*Obviously, the calls 6.3 to 6.8 yield the same result—namely the commonalities of (the approximations of) $B$ and $D$, since applied to the same set of arguments. The first and last lcs calls are in way unnecessary, too, since they extract the commonalities of copies of the same concept. It is easy to see that for complex concept descriptions $B$ and $D$ a naive realization would take substantially longer to compute the approximation of the existential restrictions than a version that reduces redundant calls in advance on every role level.*

Due to the close connection between approximation and lcs as expressed in Corollary 31 it is clear that similar examples as the one above can be constructed for the lcs as well. Example 81 does not capture redundant calls due to subsumption relationships between the elements of different tuples. Pairs of subsumer and subsumee in the same set $\mathsf{ex}_r(C)$ or in different sets $\mathsf{ex}_r(C_1), \mathsf{ex}_r(C_2)$ result in tuples that yield redundant concept descriptions and for which the computation can be omitted. However, to avoid these cases it would be necessary to, first, minimize each set $\mathsf{ex}_r(C_i)$ w.r.t. subsumption and, second, minimize the set of tuples obtained from the cross product w.r.t. to subsumption. This results in exponentially many subsumption tests. Since testing subsumption in $\mathcal{ALE}$ is NP-complete [DLN+92] (and is PSPACE-complete for $\mathcal{ALEN}$ [DLN+92], $\mathcal{ALC}$ [SS88] and $\mathcal{ALCN}$ [Hem01]), it is not advisable to reduce the redundant calls by this method. Instead we address the cases illustrated by Example 81, where redundant calls can be detected by simple syntactic comparisons. We propose the following procedure to reduce redundant calls: let $D$ be the set of tuples obtained from $\mathsf{ex}_r(C_1) \times \cdots \times \mathsf{ex}_r(C_n)$, then

1. remove duplicates in each $(C'_1, \ldots, C'_n) \in D$.
2. remove for each remaining tuple $(C'_1, \ldots, C'_n) \in D$ those tuples $(C''_1, \ldots, C''_n)$ from $D$ for which $\{C'_1, \ldots, C'_n\} \subseteq \{C''_1, \ldots, C''_n\}$.

With this procedure applied, the calls generated for concept $C$ from Example 81 are: $\mathsf{lcs}(\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B))$, $\mathsf{lcs}(\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B), \mathsf{c\text{-}approx}_{\mathcal{ALE}}(D))$ and $\mathsf{lcs}(\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D))$, where the application of the lcs is trivial for the first and last call.

In principle this kind of reduction can be performed in linear time using the results on set inclusion from [Kat04]. However, we use an unoptimized implementation of the reduction based on the built-in Common Lisp functions for sets.

Besides saving run-time by the reduction of redundant recursive calls, results also in more compact concept descriptions returned by the computation, since fewer redundant existential restrictions are generated.

### 6.2.4 Caching

An obvious idea to speed up computation is, of course, to keep and reuse results obtained from earlier runs. In our application of the common subsumers, i.e., the extension of DL knowledge bases, the knowledge engineer working on a specific subdomain of the ontology might invoke the computation of the common subsumer on the same or similar concepts in the same session. For instance, if concepts from the subdomain under consideration share similar existential restrictions, then it is likely that the same recursive calls are invoked for the computation of common subsumers for different sets of concepts. To speed up subsequent computations of common subsumers, it is desirable to keep these concept descriptions and reuse them in the construction of other common subsumers.

Even if the computation of the generalization inference is only invoked once, there might still be cases where previous results can be employed in a dynamic programming fashion. Consider Example 81 again, here obviously the results for $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(B)$ and $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(D)$ should not be recomputed for each tuple, but should be computed only once, cached and used again.

In order to use caching for generalization inferences, the main problem to be addressed is the generation of keys by which a cached value can be retrieved. The straightforward idea is to use the input concept description to index the results, for instance, on approximation. However, concept descriptions are not optimal keys for this, since, even if in normal form, they are not represented in a unique way. Thus cache misses can occur even if a result has been cached for an equivalent concept description. To remedy this effect, one would need to implement a stronger version of normal form that yields a unique representation of the concept description. To obtain such a representation would result in eliminating all redundant parts from the concept description and sort all con- and disjuncts. This is obviously not a method to generate keys for a cache in a fast way, because it generates too much overhead when storing and retrieving from the cache.

In case of the lcs (or scs) the solution for generating good keys for caching is even more complicated, since here the natural idea is to index the results by the *sets* of concept descriptions. We pursue this idea by removing duplicates from a collection of input concepts and sorting them to obtain a unique representation of the set as an input to the key generating function.

For our implementation we take a very simple approach to generating keys. We simply use the concept description as an input to the built-in Common Lisp function sxhash, that generates a compact key for the input. The return value of this function is the same, if the two inputs are the same. Thus we do not obtain the same keys

for equivalent (sets of equivalent) concepts. Nevertheless, for cases as encountered in Example 81, the obtained key is the same and thus caching based on these keys should result in a performance gain.

Our caching strategy is to cache *all* obtained results for the lcs, scs, acs and approximation in separate hash tables. This simple method is surely not feasible for very large knowledge bases and would need to be adapted for applications with this kind of knowledge bases.

Obviously there are many different caching and key generation strategies to be explored in order to optimize performance of generalization inferences. However, this kind of investigation is beyond the scope of this thesis and remains future work.

The advanced use of caching for nice concepts was mentioned in Section 6.2.2. This method reuses a cached result for a concept if another concept description that is nice is to be approximated (or when the lcs of conjunctions of nice concepts is to be computed), then the cached result can be used as a conjunct in the resulting concept description directly.

### 6.2.5 Combinations of the optimization techniques

The four optimization techniques proposed to speed up the computation of generalization inferences cannot be combined arbitrarily to gain performance. In contrast to this, some combinations enhance the efficiency of the optimizations employed. We now take a look at those combinations, where interactions can occur.

**Lazy unfolding and conjunct-wise computation.** The test whether concepts are nice requires that the concept descriptions are unfolded completely. Furthermore, the test for nice concepts cannot be carried out 'on demand' role level-wise, since the nice criterion has to hold for the whole (sub-)concept description and not just for the current role level to apply conjunct-wise computation. Thus lazy unfolding cannot be employed in combination with conjunct-wise computation to speed up generalization inferences. The normalization process, however, does not interfere with the criterion for nice concepts, thus normalization can be performed on demand, if conjunct-wise computation is employed.

**Lazy unfolding and caching.** In the case of the lcs, where using lazy unfolding can result in the use of names of defined concepts in the lcs concept description, the values to be cached can be smaller, if lazy unfolding is used. Thus this combination of optimizations can result in a better use of storage. Furthermore, equivalent concepts are easier to recognize syntactically, if concept names are not unfolded.

**Conjunct-wise computation and reduction of redundant recursive calls.** The use of conjunct-wise computation results in smaller concept descriptions handled by the approximation procedure. In turn, this can diminish the overhead for reducing redundant calls.

**Conjunct-wise computation and caching.** As pointed out earlier, the conjunct-wise computation allows to reuse cached results as conjuncts of the result concept description if the input concept is nice. Thus the number of cases where a cached result speeds up the computation of a generalization inference is increased by the use of conjunct-wise computation and detection of nice sub-concepts.

**Reduction of redundant recursive calls and caching.** The reduction of redundant recursive calls produces concept descriptions with less redundancy. On the one hand this results in smaller values to be cached and on the other hand the generation of keys is easier, if these two techniques are combined.

We implement the optimization techniques for those combinations that promise to speed up the computation of generalization inferences overall. The conjunct-wise computation is not combined with lazy unfolding in our implementation.

## 6.3 Implementation of the generalization inferences

In our implementation the representation of concept descriptions is done in a simple way. They are represented as lists in prefix notation for the concept constructors similar to the KRSS syntax [PS93]. The generalization inferences we consider in this thesis require that the input concept description(s) are unfolded and normalized. Our implementation of these steps can be used in lazy and eager mode. Unfolding for primitively defined concepts, i.e., concepts for which only necessary conditions are stated in the TBox, are unfolded by conjoining their definition to the concept name instead of replacing it.

### 6.3.1 Implementation of the LCS

Recall from Chapter 3 that the lcs is often a subprocedure for other non-standard inferences—as in our case for approximation. Thus an optimized implementation of the lcs is the groundwork for the optimized implementation of other non-standard inferences. The implementation described here is the successor of the lcs implementations for $\mathcal{ALE}$ described in [TM01; BT02a] and $\mathcal{ALEN}$ described in and [TK04b; TK04a].

We implemented the lcs computation algorithm for $\mathcal{ALEN}$ discussed in Section 3.3. This computation function is parameterized with the DL to be used. In case $\mathcal{ALE}$ is the DL to be used, the function computes the $\mathcal{ALE}$-normal form and omits the computation of induced information specific to $\mathcal{ALEN}$.

In contrast to the algorithm presented in Section 3.3, we implemented the lcs algorithm as an $n$-ary function. Taking a set of concept descriptions instead of just a pair, requires only one traversal of the concept descriptions in parallel, instead of $n-1$ ones as in the binary case. Moreover, the computation of the binary lcs, applied successively, might invoke computations of commonalities of concepts on deep role levels that do not appear in the final result. The $n$-ary lcs stops to examine deeper role levels as soon as one of the input concepts ends on the current role level.

For the computation of the lcs in $\mathcal{ALEN}$, the computation of induced information is much more involved than for $\mathcal{ALE}$. In particular, the computation of the induced number restriction $\mathsf{min}_r$ should be realized in a non-naive way. The definition of $\mathsf{min}_r$ ($\mathsf{min}_r(C) := max\{k \mid C \sqsubseteq (\geq k\, r)\}$) suggests that this number can be obtained by successive subsumption tests for increasing $k$. For $\mathsf{min}_r$ the interval to test is between the highest number mentioned in an at-least restriction and $|\mathsf{ex}_r(C)|$. We implemented the search for $\mathsf{min}_r$ by nested intervals to keep the number of subsumption tests low.[25]

The computation of the induced existential and value restrictions is realized in a straightforward way. Our implementation described in [TK04a] eliminates redundant existential restrictions of the lcs *after* they are computed, which is of course less efficient than avoiding redundant recursive calls in advance, which we do in the present implementation.

### 6.3.2 Implementation of concept approximation

Our implementation of approximation at hand is an extension of the implementation of approximation implementations described in [BKT02b] for $\mathcal{ALC}$-$\mathcal{ALE}$-approximation and in [TK04a; TK04b] for $\mathcal{ALCN}$-$\mathcal{ALEN}$ approximation.

The optimization techniques mentioned in the last section aside, our implementation is again a straightforward one. If a less expressive DL than $\mathcal{ALEN}$ (or $\mathcal{ALE}$) is chosen as a target DL, then, instead of removing sub-concept descriptions using the undesired constructors afterwards, the target DL is taken into account directly. This results in computation of induced information for all constructors of the source DL, but no recursive call for concept constructors not supported in the target DL.

The approximation implementation uses the lcs implementation just described for the recursive calls and computation of induced information, for instance to obtain $\mathsf{ind\text{-}val}_r$. Thus, approximation should also benefit from the optimizations for the lcs. In case caching is employed for approximation, we also use caching for the lcs.

#### An improved computation algorithm

Strictly speaking, the replacement of lcs calls by disjunction for the computation of induced information of $\mathcal{ALCN}$-concept descriptions already is an optimization, compared to the approach pursued in [BKT02a]. One can carry this replacement further by the use of Corollary 31 and use the equivalence

$$\mathsf{approx}_{\mathcal{L}_2}\Big(\bigsqcup_{1\leq i\leq n} C_i\Big) \;\equiv\; \mathsf{lcs}_{\mathcal{L}_2}(C_1,\ldots,C_n)$$

to avoid the lcs calls during the computation of approximation by simply making use of disjunction. More precisely, the calls $\mathsf{lcs}_{\mathcal{L}_2}\big(\{\mathsf{approx}_{\mathcal{L}_2}(E_i) \mid 1 \leq i \leq n\}\big)$ can be replaced by $\mathsf{approx}_{\mathcal{L}_2}(\bigsqcup_{1\leq i\leq n} E_i)$. Applying Corollary 31 to the expressions for the

[25] This might seem to gain only little runtime, since most knowledge bases we encountered only use relatively small numbers in number restrictions. However, if for example, number of seats in airbusses are modeled, this procedure might pay.

Input: $\mathcal{ALCN}$-concept description $C$.
Output: $\mathcal{ALEN}$-approximation $D$ of $C$.

1. If $C \equiv \bot$, then $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}(C) := \bot$
2. If $C \equiv \top$, then $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}(C) := \top$
3. Otherwise, transform $C$ into $\mathcal{ALCN}$-normal form and return

$$
\begin{aligned}
&\mathsf{d\text{-}approx}_{\mathcal{ALEN}}(C) := \\
&\quad \sqcap_{A \in \bigcap_i prim(C_i)} A \\
&\quad \sqcap \; (\geq \;\; \min\{\mathsf{min}_r(C_i) \mid 1 \leq i \leq n\}\; r) \\
&\quad \sqcap \; (\leq \;\; \max\{\mathsf{max}_r(C_i) \mid 1 \leq i \leq n\}\; r) \\
&\quad \sqcap \bigsqcap_{\substack{(C'_1,\ldots,C'_n) \in \\ \mathsf{ind\text{-}ex}_r(C_1) \times \cdots \times \mathsf{ind\text{-}ex}_r(C_n)}} \mathsf{d\text{-}approx}_{\mathcal{ALEN}}\big(\textstyle\bigsqcup_{1 \leq i \leq n} C'_i \sqcap \mathsf{val}_r(C_i)\big) \\
&\quad \sqcap \forall r.\mathsf{d\text{-}approx}_{\mathcal{ALEN}}\big(\textstyle\bigsqcup_{1 \leq i \leq n} \mathsf{ind\text{-}val}_r(C_i)\big)
\end{aligned}
$$

Figure 6.4: The disjunction-based algorithm $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}$.

recursive call of the $\mathcal{ALEN}$-approximation algorithm yields:

$$\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{approx}_{\mathcal{ALEN}}(E_i) \mid 1 \leq i \leq n\}\big) \;\equiv\; \mathsf{approx}_{\mathcal{ALEN}}\big(\bigsqcup_{1 \leq i \leq n} \mathsf{approx}_{\mathcal{ALEN}}(E_i)\big).$$

Since by definition of approximation $E_i \sqsubseteq \mathsf{approx}_{\mathcal{ALEN}}(E_i)$ holds, and disjunction and approximation are both generalization operations, we obtain:

$$\mathsf{approx}_{\mathcal{ALEN}}\big(\bigsqcup_{1 \leq i \leq n} E_i\big) \;\sqsubseteq\; \mathsf{approx}_{\mathcal{ALEN}}\big(\bigsqcup_{1 \leq i \leq n} \mathsf{approx}_{\mathcal{ALEN}}(E_i)\big).$$

Thus the replacement of the expression $\mathsf{lcs}_{\mathcal{ALEN}}\big(\{\mathsf{approx}_{\mathcal{ALEN}}(E_i) \mid 1 \leq i \leq n\}\big)$ by $\mathsf{approx}_{\mathcal{ALEN}}(\bigsqcup_{1 \leq i \leq n}^{n} E_i)$ does not yield a more general concept description. Clearly, $\bigsqcup_{1 \leq i \leq n} E_i \sqsubseteq \mathsf{approx}_{\mathcal{ALEN}}\big(\bigsqcup_{1 \leq i \leq n} E_i\big)$. Thus we obtain an alternative way to compute $\mathcal{ALE}$-approximations. To make this procedure more precise, we display $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}$, the algorithm for $\mathcal{ALEN}$-approximation of $\mathcal{ALCN}$-concept descriptions obtained by this replacement in Figure 6.4. It is clear that by the same kind of replacement one can obtain a version of the computation algorithm for $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concept descriptions that does use disjunction instead of computing the lcs.

The algorithm $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}$ does not yield a better method in terms of computational complexity, since it trades exponentially many lcs computations (of approximations of concepts bounded by the size of $C$) for a single approximation computation of exponentially many disjuncts bounded by the size of $C$. However, in practice, we would expect that $\mathsf{d\text{-}approx}_{\mathcal{ALEN}}$ shows better performance, since it only considers

existential and value restrictions common to all disjuncts, while c-approx$_{\mathcal{ALEN}}$ might compute the approximation of some existential or value restriction of some $C_i$ on some role level $l$ that is not considered later in the computation of the lcs since some other approximations of $C_j$ end on role level $n \leq l$. Thus the possibly costly computation of the approximation of the last $l - n$ role levels is wasted in c-approx$_{\mathcal{ALEN}}(C_i)$. We have not yet implemented this version of the approximation algorithm.

### 6.3.3 Implementation of the good common subsumers acs and scs

In case of common subsumers we typically extend a background terminology by a user terminology. Thus, in principle, two TBoxes come into play. In our current implementation we do not distinguish between the two TBoxes, but use the union of their definitions as the underlying TBox. This requires that unfolding checks if the definition of a concept uses concept constructors that are not supported by the user DL and if so, unfolding leaves the concept name in the concept description instead of replacing it by its definition. In case that $\mathcal{ALE}$-unfolding is used (according Definition 67 on page 99), it is checked if the concept definition containing DL constructors not from the user DL can be converted into a user DL concept descriptions by pushing negation inwards. By this procedure more unfolding steps can be performed, which in turn make more information form the background TBox explicit. The experiments in [BST07] showed that $\mathcal{ALE}$-unfolding results in more specific common subsumers in a significant percentage of of cases.[26]

We have introduced three kinds of computation methods for obtaining good common subsumers in Section 5.4. We focus here on the implementation of the scs and the acs. The implementation of the third method can be found discussed in detail in the thesis of Sertkaya [Ser07].

In case of the approximation-based good common subsumer (acs) the implementation is very simple. First, $\mathcal{ALE}$-unfolding has to be performed on each of the input descriptions. Next, we build their disjunction as the input to the approximation function. We use the approximation function which translates to the DL offering the same set of concept constructors as the user DL.

In case of the common subsumer based on subsumption closure (scs) our implementation also starts by performing $\mathcal{ALE}$-unfolding for the input concept descriptions. The actual scs function is in fact obtained from the lcs function by changing the way (negated) concept names are treated. Instead of the intersection of (negated) concept names, the intersection of the subsumption closures of the (negated) names is used. For the computation of the subsumption closure of the concept names, we use the function concept-ancestors and concept-descendants offered by most DL reasoners. The function concept-ancestors (concept-descendants) retrieves the named concepts that subsume (are subsumed by) a given input concept from the concept hierarchy.

As the lcs, the scs is also implemented as an $n$-ary operator. Furthermore the same optimizations as for the lcs are implemented for this inference.

[26] The experiments in [BST07] resulted on the average in more specific common subsumers in about 12 % of the cases when $\mathcal{ALE}$-unfolding was used.

## 6.4 Implementation of heuristics for syntactic algorithms

We now turn to the second group of non-standard inferences devised or discussed in this thesis: the difference operator for computing the difference between $\mathcal{ALC}$- and $\mathcal{ALE}$-concept description and the computation algorithm for minimal rewritings in $\mathcal{ALE}$. Compared to the implementation of the generalization inferences, these implementations do not employ particular optimization techniques.

### 6.4.1 Implementation of the difference operator

We implemented the improved heuristic of the algorithm c-diff for computing the difference of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions as displayed in Figure 4.3 on page 80. The improved heuristic uses $\mathsf{c\text{-}diff}(E, \mathsf{c\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_r(C) \sqcap \mathsf{val}_r(D))$ instead of the call $\mathsf{c\text{-}diff}(E, \mathsf{val}_r(C) \sqcap \mathsf{val}_r(D))$ in the last line of the c-diff algorithm.

We need the difference operator mainly to assess the information loss caused by approximating a concept description. In this application of the difference operator the original and the approximated concept description are already unfolded. Of course in the general case, lazy unfolding and normalization are employed for this algorithm also, since it proceeds role level-wise, too.

Implementation- and performance-wise the interesting part of the c-diff is the reduction of the disjunction (Step 2 in c-diff) and the reduction of each disjunct of the $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form (Step 3 in c-diff). In order to implement the reduction of the disjunction is realized exactly as described in c-diff; generating a quadratic number of subsumption tests in the number in disjuncts. The reduction of the set $\mathsf{ex}_r$ is realized such that the most selective testing condition is applied first. In the case of assessing the difference of a concept $C$ and its approximation $D$ it is more likely that there are more concept descriptions in $\mathsf{ex}_r(C)$ that are redundant to the concept descriptions from $\mathsf{ex}_r(D)$ than redundant concept descriptions in $\mathsf{ex}_r(C)$. Thus we test Condition (ii) from c-diff before we test Condition (i) to reduce $\mathsf{ex}_r(C)$.

### 6.4.2 Implementation of minimal rewriting

We only introduced the non-standard inference minimal rewriting briefly in Section 2.4.3. In a nutshell, the idea of the computation of a minimal rewriting w.r.t. a TBox is to re-introduce names of defined concepts from the TBox in exchange for complex sub-concept descriptions. Thus the effect of computing a minimal rewriting is somewhat inverse to unfolding a concept description.

An algorithm to compute minimal rewritings of $\mathcal{ALE}$-concept descriptions w.r.t. unfoldable $\mathcal{ALE}$-TBoxes has been devised in [BKM00]. The complexity of the proposed algorithm is in NP with an oracle for subsumption. To obtain better performance in practice the authors proposed a heuristic in [BKM00] that can compute (not necessarily minimal) rewritings in polynomial time, again with an oracle for subsumption.

The algorithm to compute minimal rewritings for $\mathcal{ALE}$-concept descriptions as well as the heuristic for it proceed in two phases:

1. Extension phase: (negated) concept names that subsume sub-concept descriptions of the input $\mathcal{ALE}$-concept description are conjoined to those sub-concept descriptions.

2. Reduction phase: redundant complex concept descriptions are removed from the concept description recursively.

The heuristic algorithm performs these phases interlaced and proceeds by a greedy heuristic. In the reduction phase the conjunction of names of defined concepts obtained from the extension phase are copied, unfolded and stored as the so-called *context*. The reduction phase removes sub-concept descriptions that are redundant to this context. This phase basically performs steps that are very similar to the difference operator. In order to obtain a polynomial procedure, it is important to use lazy unfolding for the context by this rewriting procedure.

Interestingly, this procedure can be employed for obtaining smaller rewritings for $\mathcal{ALE}(\mathcal{T})$-concept descriptions as well, since the first phase only conjoins concept names or their negation (which possibly stand for $\mathcal{ALC}$-concepts, if defined in the background terminology) and the second phase only removes parts of the concept description. Thus no concept constructors used in the background TBox that are not covered by the user DL, are added to the concept description.

In our application of extending DL knowledge bases the rewriting implementation would be employed to compute small representations of $\mathcal{ALE}$-concept descriptions obtained by the lcs, approximation, acs or scs.

## 6.5 The non-standard inference system SONIC

The non-standard inference system SONIC[27] implements a collection of non-standard inferences and offers these inferences to users by plug-ins for ontology editors. More precisely, SONIC consists of the SONIC server and the SONIC front-end, both of which we describe in more detail in the following.

### 6.5.1 The SONIC server

The server implements the inferences discussed in this thesis and the programming interfaces for their use. In particular, SONIC comprises the implementations of inferences described in this chapter and concept matching for $\mathcal{ALE}$ (for a description of the concept matching algorithm for $\mathcal{ALE}$ see [BK00]). The implementation of this inference was done by Brandt and is described in [Bra03; Bra06]. By means of this implementation SONIC is also capable of supporting 'extension by modification' for DL knowledge bases, that we described in Section 1.2.2, to some extent.

Furthermore, the SONIC server implements an approach to obtain the hierarchy of least common subsumers without computing the actual lcs concept descriptions. This approach was initially described in [BM00]. The idea is that the modeler picks a collection of concepts $C_1, \ldots, C_n$ that promise to yield a lcs that extends the concept

[27] SONIC is an acronym for simple ontology non-standard inference component.

hierarchy in a desired way. Then, by attribute exploration from concept analysis, the subsumption lattice of the lcs concepts of all subsets of $\{C_1, \ldots, C_n\}$ is computed. This is an interactive method that has to ask some subsumption relations of the elements from the (prospective) lattice during computation. In our case these questions are of the form: 'Does the lcs of $C_{i_1}, \ldots, C_{i_m}$ subsume the lcs of $C_{j_1}, \ldots, C_{j_k}$?', where the arguments are from the set of concepts selected by the user. To answer this kind of questions the lcs computation function is invoked. However, attribute exploration asks a minimal number of these questions. Moreover, some of these questions can be answered by exploiting subsumption or subset relations between the arguments from the least common subsumers in question—as for example in 'Does the lcs of $C_1, C_2, C_3, C_4$ subsume the lcs of $C_1, C_2$?', where the answer obviously is 'yes'. Once the subsumption lattice is obtained and displayed to the modeler, she can see whether the set of picked concepts contains a concept that causes the lcs to be $\top$. Based on this lattice the choice of a good subset of lcs arguments can be made without having to compute all (or many) of them by individual lcs calls.

The Sonic server needs to be connected to a standard DL reasoner in order to compute inferences. Moreover, if Sonic is used via the Sonic ontology editor plug-in, then it is advantageous to connect to the same instance of the reasoner that the ontology editor uses also in a three-tier fashion. In such a setting the ontology only needs to be stored and classified in one reasoner. To this end Sonic is equipped with several interfaces. Sonic supports the DIG 1.0 interface [BMC03] and a TCP-based interface (LRacer [Rac05]) to RacerPro. If the Sonic server is to be used without the plug-in, then this three-tier architecture is unnecessary and then, since also being a common lisp system, RacerPro can be used directly via in-process lisp calls. This last setting shows much better performance because of the many queries Sonic has to make to the standard reasoner.[28]

In the next version Sonic will support the DIG 2.0 interface [TBK+06]. On the one hand this interface can be used for connecting to to DIG 2.0 compliant DL reasoners. On the other hand, and more importantly, Sonic will support the non-standard inference extension of the DIG 2.0 interface [TBK+06]. By means of this extension other applications will be able to use the Sonic reasoning services.

### 6.5.2 The Sonic front end

This part of Sonic supplies a GUI for employing non-standard inferences from within an ontology editor. The Sonic front end of the first release of Sonic [TK04a; TK04b] was tailored to the ontology editor OilEd [BHGS01]. Later the ontology editor Protégé [GMF+03] became more popular and the de-facto standard ontology editor of today. This editor can be extended with different plug-ins. From the DL perspective the OWL plug-in [KMR04] is the most notable one, since it supports the editing of and reasoning with OWL ontologies by connecting to a DIG compliant DL reasoner. The more recent versions of Sonic [Tur05] provide a plug-in for Protégé only.

[28] Tests have shown that Sonic, if the DIG or the TCP interface is used, spends about 80 % of its computation time for communicating with the DL reasoner.

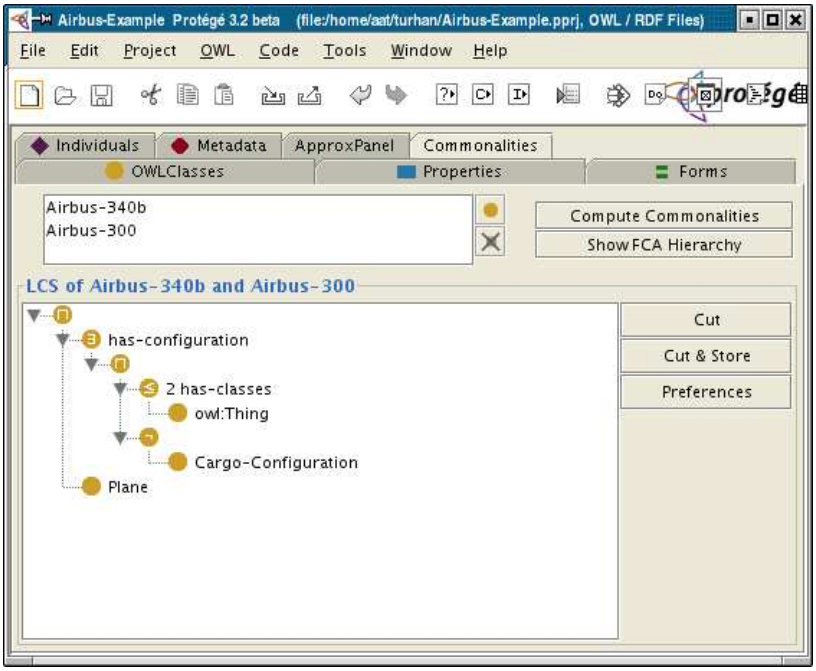


Figure 6.5: SONIC's commonalities panel.

The SONIC front end is written in Java and implements the GUI of the plug-in and the interface to the SONIC server. It provides four panels that can be activated individually:

**The approximation panel** supplies access to the concept approximation implementation. This panel allows the user to select a concept name from the ontology and invoke approximation for this concept. The resulting concept description is then displayed as a syntax tree to the user. This syntax tree can be edited by removing sub-concept descriptions from the tree. Furthermore, the concept description can be assigned a name and be added to the ontology as a new concept definition.

**The commonalities panel** supplies computation of commonalities by the lcs or the scs. The panel is displayed in Figure 6.5. The user can set the option which of the two inferences should be employed. Furthermore, it can be chosen under preferences, whether the result of the inference should be compressed by rewriting. To compute commonalities the user can select a set of concept names from the list of all concepts present in the ontology to invoke the computation. Similar to the approximation panel, the resulting concept description is displayed as a syntax tree as can be seen in Figure 6.5. The user can remove unwanted sub-concept descriptions from this concept description, can assign a name to the obtained concept description and add it to the ontology.

Furthermore the commonalities panel supports the choice of interesting concept sets to be generalized by the lcs. Here the user can select a set of concept names for which the lcs is of interest. SONIC displays the subsumption lattice of all subsets of

the set of selected concepts as a graph by the use of the Grappa Java Graph Package [Res]. Then the modeler can invoke the computation of the lcs concept description for interesting subsets of the selected concepts.

**The matching panel** of the SONIC front end allows to access the service to compute matchers based on the $\mathcal{ALE}$-matching algorithm implementation described in [Bra03; Bra06]. Here, the user can enter a concept pattern for which the matchers present in the ontology are computed by the SONIC server. To rule out an overlap of the sets of concept names and variable names, the SONIC matching panel offers a preset set of five variables to formulate a concept pattern. The matching concepts from the ontology for the specified concept pattern are displayed in a list to the user.

In its current form the SONIC front end provides support for the following methods to extend DL knowledge bases:

- the bottom-up extension by offering the lcs, approximation and rewriting,
- the customization of background knowledge bases by the scs, and
- the extension by modification by finding matchers.

In case disjunction is used in the knowledge base, the modeler first invokes the computation of concept approximation of the concepts for which a non-trivial lcs is desired and then invokes the computation of the lcs to obtain a meaningful lcs, that can be edited.

The GUI of the SONIC front end is clearly a prototype. It is not tailored to the process of any of the extension approaches. To this end an 'extension wizard' that guides the user through the process of coming up with a suitable concept description for a new definition and uses the inferences depending on the DL(s) employed competently might be more helpful for naive users than the plain panels. However, the usefulness of the inferences for the proposed tasks can be demonstrated by the SONIC front end in its current form.

With the described implementations at hand the implementation of the generalization inferences in the SONIC server are no longer just a prototype implementation, but can count for optimized implementations despite the fact that they are using a simple representation of concept descriptions. Furthermore, our implementation provides a good test-bed for experiments to optimize these inferences further. Before focussing on the performance of the inferences, one should examine the quality of the proposed approaches, which we do in the next chapter.

# Chapter 7

# Evaluation of the common subsumer approaches

In this chapter we present a comparative evaluation of the approaches to obtain non-trivial common subsumers developed in this thesis. The underlying inferences to support the computation of common subsumers as well as the inferences to complement the approaches are implemented and are evaluated in regard of their usefulness with respect to their application: the extension of DL knowledge bases. This implies for our evaluation that the test data should be from practical applications than randomly generated. Next, we describe the test data employed in our tests and then we discuss the experiments and their outcome.

## 7.1 The test data

We test our implementation of the inferences on concept definitions stemming from two knowledge bases used in practical applications. In contrast to automatically generated test data this input knowledge bases is more similar to the concept descriptions used in our application scenario for the inferences that we want to test. Randomly generated input data tends to highlight effects not necessarily appearing in practice. Furthermore, the structure of the knowledge base and its concept definitions influences strongly the outcome of the generalization inferences. For instance, if the TBox contains a lot of redundancy in the concept definitions, the resulting concept descriptions will do so, too. The usefulness of conjunct-wise approximation, for example, cannot be assessed by randomly generated data, since we would need to know whether and if, how often nice concept descriptions appear in practical knowledge bases.

For our test of the computation of common subsumers w.r.t. a background terminology, we assume an empty user TBox and thus can use the same underlying TBoxes as for the tests for the approximation-based approach. In the following we introduce the two knowledge bases used in our experiments.

### The DICE knowledge base

The DICE [29] TBox, is a medical knowledge base from the intensive care domain modeling reasons for admission in intensive care, see [CAH03; CAH04]. The DICE knowledge base has been developed at the department of Medical Informatics at the Academic Medical Center in Amsterdam since the late 1990s.

This knowledge base models 1456 reasons for admission that comprise both diseases and procedures that require intensive care and monitoring of patients. It contains 34 roles and about 3500 concepts, of which 3249 have a (in most cases primitive) definition. These definitions yield an unfoldable TBox—acyclic with (primitive) concept definitions only. Originally, this TBox uses the DL $\mathcal{ALCQ}$ and disjointness statements. We use a variant of the DICE TBox for our testing.

DICE$_{\mathcal{ALC}}$ is the TBox obtained from the DICE TBox by removing the disjointness statements from the TBox and and by deleting the sub-concept descriptions that are (qualified) number restrictions.

We want to employ the syntactic operators for our evaluation and since they are only available for $\mathcal{ALE}$, we do not use number restrictions in our tests.

### The OntoCAPE knowledge base

The OntoCAPE [30] knowledge base is developed at the Chair for Process Systems Engineering at RWTH Aachen University. It models concepts from the process engineering domain, such as chemical reactions, parts of chemical plants and also mathematical models to describe the behavior of the systems. The OntoCAPE knowledge base is designed in the layered fashion that we mentioned in Section 1.1.3, see also [MYM07]. It contains 575 (in most cases primitive) concept definitions.

Originally, this TBox uses the DL $\mathcal{SHIQ}(\mathcal{D})$, i.e., concept constructors from $\mathcal{ALCQ}$ in combination with data types, role declarations for inverse roles and transitivity and domain and range restrictions for roles and attributes. Moreover, it contains GCIs and cyclic definitions. Again, we use a variant of the OntoCAPE TBox that use only concept constructors the generalization inferences and the syntactic operators can handle:

OntoCAPE$_{\mathcal{ALC}}$ is the TBox obtained from the OntoCAPE TBox by removing the GCIs, role declarations, data-type statements and deleting sub-concept descriptions that are number restrictions. Furthermore, we changed two concept definitions by removing sub-concept descriptions, such that an acyclic TBox was obtained.

### The test concept sets for the computation of common subsumers

To evaluate the approaches for the computation of common subsumers in presence of disjunction and the inferences that realize them, we need sets of concepts that

[29] DICE is an acronym for: Diagnoses for Intensive Care Evaluations.

[30] The name OntoCAPE stems from ONTOlogy-based Computer Aided Process Engineering.

| | nr. of concepts | nr. of definitions | nr. of test tuples |
|---|---|---|---|
| $\mathsf{DICE}_{\mathcal{ALC}}$ | 3500 | 3249 | 75 |
| $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ | 588 | 575 | 60 |

Table 7.1: Test data overview.

we can use as input for the common subsumer inferences in our tests. We selected these input sets by first classifying the test ontology and then identifying concepts with many concept children, i.e., direct subsumees. From this set of direct subsumees we picked subsets randomly, which are the input test data for our evaluation of the common subsumer inferences.

The idea behind this way of selecting the input is to simulate the application of the bottom-up approach, where unbalanced concept hierarchies are augmented with new concepts to obtain a more tree-like concept hierarchy by introducing a new parent concept for sibling concepts. So, by identifying concepts with many concept children, we focus on a part of the concept hierarchy that a knowledge engineer might select for an extension by an intermediate concept. Moreover, by this way of selecting the input sets, we guarantee that no trivial common subsumers are obtained, that collapse to $\top$. We would always obtain at least the common parent concept—thus the computation of common subsumers of our test sets is not completely trivial.

We randomly picked 75 such concept sets in the above described fashion from $\mathsf{DICE}_{\mathcal{ALC}}$ and 60 such sets from $\mathsf{OntoCAPE}_{\mathcal{ALC}}$. Each of the sets contains 2 to 7 concept names.

**Test environment**

We ran our tests on a standard PC with 500MB of memory under Linux. The Lisp source code for the inferences was compiled and ran under Allegro Common Lisp 8.0. As the underlying standard reasoner we used RacerPro (Version 1.9.1) also compiled under Allegro Common Lisp 8.0, where the calls to the reasoner were performed as in process calls.

## 7.2 Evaluation of the precision of common subsumers

To evaluate the usefulness of the concept descriptions obtained by the inferences for computing common subsumers, we concentrate on the precision of the obtained result. Here, precision is to be understood in terms of information loss between the disjunction of the input concepts—the trivial least common subsumer—and the concept description obtained by applying one of the techniques for the 'meaningful common subsumer'. We proceed by evaluating the precision of the two approaches individually.

To assess the precision of the concept descriptions obtained by the common subsumer we proceed by testing whether the trivial lcs, i.e., the disjunction of the input concepts, is equivalent to the common subsumer obtained by the approximation-based approach, the scs or the acs. If not, we can estimate which information was lost for the approximation-based approach in a second step by computing the difference for

the unfolded trivial lcs and the approximation-based lcs, i.e., the concept description obtained by first approximating each input concept description in $\mathcal{ALE}$ and then computing their lcs. Since the difference operator as introduced in Section 4.3 was devised for the syntactic difference of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions, we use the $\mathcal{ALC}$-ontologies $\mathsf{DICE}_{\mathcal{ALC}}$ and $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ and the concept sets of defined concepts obtained from these two TBoxes for this kind of tests.

In our setting the acs yields equivalent concept descriptions to the ones obtained by the approximation-based approach. The acs is obtained by unfolding the $\mathcal{ALE}(\mathcal{T})$-input concepts completely, yielding $\mathcal{ALC}$-concept descriptions and then applying the $\mathcal{ALE}$-approximation to the disjunction of the unfolded input concept descriptions. The equivalence

$$\mathsf{approx}_{\mathcal{ALE}}\Big(\bigsqcup_{1\leq i\leq n} C_i\Big) \;\equiv\; \mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{approx}_{\mathcal{ALE}}(C_i) \mid 1 \leq i \leq n\}\big)$$

is the one we used to devise $\mathsf{d\text{-}approx}_{\mathcal{ALE}}$. Although the concept descriptions obtained by the two methods need not be the same syntactically, the evaluation of the approximation-based approach based on the difference operator carries over to the acs to some extent.

For the subsumption closure-based common subsumer (scs) computed w.r.t. a background terminology we cannot use the difference operator to assess the information loss, in case the common subsumer obtained by these methods is more general than the trivial lcs. To see the reason for this, consider the following example where the TBox is $\mathcal{T} = \{A = B \sqcup C\}$ and we are interested in the lcs of $C_1 = B \sqcap \exists r.D$ and $C_2 = C \sqcap \exists r.E$. Then the $\mathsf{lcs}_{\mathcal{ALC}}(C_1, C_2) = C_1 \sqcup C_2$, while the $\mathsf{scs}_{\mathcal{ALE}(\mathcal{T})}(C_1, C_2) = A \sqcap \exists r.D \sqcap \exists r.E$. Both concept descriptions are equivalent, but the difference operator would return the syntactic difference between them. In this case the syntactic difference is misleading to assess the information loss. Applying the difference operator to the unfolded concept description obtained by the scs is not possible either, since it is an $\mathcal{ALC}$-concept descriptions for which we would need a difference operator that can compute the syntactic difference between $\mathcal{ALC}$-concept descriptions.

### 7.2.1 Precision of the approximation-based approach

To evaluate the precision of the approximation-based approach, we computed for each concept set $S = \{C_1, \ldots, C_n\}$ in the test data the following concept descriptions:

1. the trivial lcs: the disjunction of the concepts in the concept set unfolded w.r.t. the underlying TBox: $\mathsf{lcs}_{\mathcal{ALC}}(C_1, \ldots, C_n) = \mathsf{unfold}(\bigsqcup_{1\leq i\leq n} C_i)$.

2. the approximation-based lcs: the $\mathcal{ALE}$-lcs of the set of $\mathcal{ALE}$-approximations of each $\mathcal{ALC}$-concept from the concept set:
   $\mathsf{lcs}_{\mathsf{approx}}(C_1, \ldots, C_n) = \mathsf{lcs}_{\mathcal{ALE}}\big(\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C_i) \mid 1 \leq i \leq n\}\big)$.

3. the syntactic difference of the trivial lcs and the approximation-based lcs:
   $D_{\mathsf{approx}} = \mathsf{diff}(\mathsf{lcs}_{\mathcal{ALC}}(C_1, \ldots, C_n),\ \mathsf{lcs}_{\mathsf{approx}}(C_1, \ldots, C_n))$.

| | $\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{lcs}_{\mathsf{approx}}$ |
|---|---|
| $\mathsf{DICE}_{\mathcal{ALC}}$ | 36 (48,0%) |
| $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ | 8 (12,3%) |

Table 7.2: Comparison of $\mathsf{lcs}_{\mathcal{ALC}}$ and $\mathsf{lcs}_{\mathsf{approx}}$.

| | $\lvert\mathsf{lcs}_{\mathcal{ALC}}\rvert$ | $\lvert\mathsf{lcs}_{\mathsf{approx}}\rvert$ | $\lvert\mathsf{diff}(\mathsf{lcs}_{\mathcal{ALC}}, \mathsf{lcs}_{\mathsf{approx}})\rvert$ | $\mathsf{diff}(\ldots) \equiv \top$ |
|---|---|---|---|---|
| $\mathsf{DICE}_{\mathcal{ALC}}$ | 68,1 | 7,4 | 14,7 | 39 (52,0%) |
| $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ | 32,2 | 2,2 | 15,4 | 50 (83,3%) |

Table 7.3: Applying the difference operator to $\mathsf{lcs}_{\mathcal{ALC}}$ and $\mathsf{lcs}_{\mathsf{approx}}$.

We ran these tests for the $\mathsf{DICE}_{\mathcal{ALC}}$ and the $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ test data. Table 7.2 shows in the first column the number of cases where the trivial lcs is strictly more specific than the concept description obtained by the approximation-based approach. In these cases information captured in the trivial lcs, common to all input concept descriptions was lost when computing the lcs of the $\mathcal{ALE}$-aproximations of the concept descriptions. It shows that information is lost in 48% of the cases tested for the $\mathsf{DICE}_{\mathcal{ALC}}$ TBox and 12,3% for the $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ TBox.

The Table 7.3 shows the average concept size of the trivial lcs, of the approximation-based lcs, and of their difference obtained by the heuristic for computing the difference. It shows for both test TBoxes that the difference between the $\mathsf{lcs}_{\mathcal{ALC}}$ and the $\mathsf{lcs}_{\mathsf{approx}}$ results in concept descriptions a couple of times larger that the $\mathsf{lcs}_{\mathsf{approx}}$ itself. This might seem a daunting result at first, but recall that the heuristic for computing the difference applied to concept description with redundancy yields a syntactic difference with redundancies. In fact, we obtained a difference equivalent to $\top$ in the majority of the cases for both TBoxes (see last column). Thus the concept sizes for the difference give a biased picture of the quality of $\mathsf{lcs}_{\mathsf{approx}}$.

In 75 of the test cases for the $\mathsf{DICE}_{\mathcal{ALC}}$ TBox we obtained a concept name as the result of applying the approximation-based approach, which indicates that the common parent concept of the concepts from the tuple was obtained as their common subsumer. For the OntoCAPE knowledge base 11 such cases were found. In regard of our application scenario these are the cases where no new node is introduced in the concept hierarchy and the modeler would have to revise her choice of input concepts.

### 7.2.2 Precision of the common subsumers for background ontologies

For the evaluation of the precision of the common subsumers computed for the customization of background terminologies we examine the same quality criteria as above for the computation of the scs and the acs, for an evaluation for the gcs as introduced in Section 5.4.1, see [Ser07].[31] To assess the precision in this setting, we can only refer

[31] The somewhat discouraging results in [BST07] regarding the sizes of TBoxes that can be handled by the FCA based method are superseded by the results in [Ser07]. In the forthcoming thesis of Sertkaya an improvement by two orders of magnitude is achieved for the number of expert calls the

| | $\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{scs}$ | $\mathsf{lcs}_{\mathcal{ALC}} \sqsubset \mathsf{acs}$ | $\mathsf{scs} \sqsubset \mathsf{acs}$ |
|---|---|---|---|
| $\mathsf{DICE}_{\mathcal{ALC}}$ | - | 36 (48,0%) | 36 (48,0%) |
| $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ | 2 ( 3,1%) | 8 (12,3%) | 52 (80,0%) |

Table 7.4: Subsumption relationships between $\mathsf{lcs}_{\mathcal{ALC}}$, acs and scs.

to the subsumption relationships between the obtained concept descriptions, since the difference operator does not yield meaningful results in this setting for the reasons explained earlier. We computed for each concept set $S = \{C_1, \ldots, C_n\}$ in the test data:

1. the trivial lcs: the disjunction of the concepts in the concept set unfolded w.r.t. the underlying TBox: $\mathsf{lcs}_{\mathcal{ALC}}(C_1, \ldots, C_n) = \mathsf{unfold}(\bigsqcup_{1 \leq i \leq n} C_i)$.

2. the approximation-based gcs: the acs of the disjunction of $\mathcal{ALE}(\mathcal{T})$-concepts from the concept set:
$\mathsf{acs}(C_1, \ldots, C_n) = \mathsf{c\text{-}approx}_{\mathcal{ALE}}(\bigsqcup_{1 \leq i \leq n} C_i)$

3. the subsumption closure-based gcs: the scs of the concept set:
$\mathsf{scs}(C_1, \ldots, C_n)$

We computed these concept descriptions for the tuples from the $\mathsf{DICE}_{\mathcal{ALC}}$ and the $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ test data. We checked for the subsumption relations between the obtained concept descriptions. The results are displayed in Table 7.4. The first two columns show the number of cases, where scs (acs) is more general than the trivial lcs. It shows, that the scs is only in two cases more general than the lcs and thus does result in hardly any information loss for our test data.

For the acs, we obtain the same information loss, as for the approximation-based approach. Interestingly, the scs results always in a more specific concept description than the acs, if the two are not equivalent. This is somewhat different from the results in [BST07], where also cases appeared in which the acs was more specific than the scs.

In this setting we obtained only 8 trivial acs concept descriptions, i.e., concepts that collapsed to the common parent concept. For the scs this number of collapsed concepts is 2. Regarding the precision of common subsumers, the scs showed the best performance on our test data.

## 7.3 A look at the performance

In Table 7.5 we see the average run-times measured for the different ways to obtain common subsumers. These run-times were obtained by using an implementation that realizes lazy unfolding. It shows that with this optimization technique applied alone the run-times for our examples from practical applications are in most cases below 1,5 seconds. This is already an acceptable run-time for interactive use—where run-time

FCA based method needs, which reduced the overall run-time drastically.

| | $\mathsf{lcs}_{\mathcal{ALC}}$ | $\mathsf{lcs}_{\mathsf{approx}}$ | acs | scs |
|---|---|---|---|---|
| $\mathsf{DICE}_{\mathcal{ALC}}$ | 1,34 | 6,52 | 1,39 | 0,23 |
| $\mathsf{OntoCAPE}_{\mathcal{ALC}}$ | 0,15 | 0,29 | 0,15 | 0,03 |

Table 7.5: Average run-times of $\mathsf{lcs}_{\mathcal{ALC}}$, $\mathsf{lcs}_{\mathsf{approx}}$, acs and scs (in $s$).

measured for $\mathsf{lcs}_{\mathsf{approx}}$ might be seen as an exception. However, it is, again the scs that shows the best performance, when comparing the three approaches to obtain non-trivial common subsumers. It only uses a fraction of the run-times of the other common subsumers. Surprisingly, the computation of the scs is even faster than the trivial lcs. This effect is due to $\mathcal{ALE}$-unfolding, where the input concept description is not necessarily unfolded completely (if concept definitions are encountered that cannot be transformed into an $\mathcal{ALE}$-concept by De Morgan' rules), while the trivial lcs is obtained by unfolding the disjunctions of the input concepts completely.

The results also indicate that concept approximation is the inference that would benefit most from an optimized implementation. A through evaluation of the performance gain of the individual techniques discussed in Section 6.2 and their possible, meaningful combinations is out of scope of this thesis. Nevertheless, we would like to pick up a question raised in Section 6.2.2 here. In order to be able to apply conjunct-wise computation for approximation, we have to test whether a concept description is nice. Although the conditions for this test have been relaxed in Section 6.2.2, the question still is: do the concept definitions from application knowledge bases contain nice concepts? An investigation of the DICE and the OntoCAPE knowledge base showed that nice concepts do appear in knowledge bases from applications [TB07]. In case of the DICE knowledge base, 13,2% of the concepts are nice. The OntoCAPE knowledge base contains even about 35% of nice concepts. Thus conjunct-wise approximation might help to obtain better run-times for approximation computed w.r.t. knowledge bases obtained from practical applications.

Our evaluation of the common subsumer approaches showed that the common subsumers obtained by applying the lcs to the concepts obtained by approximation performs well. In more than half of the cases the approximation-based approach captures the full information of the trivial lcs. Similarly, the here proposed approaches for the computation of common subsumers w.r.t. a background knowledge base performed well w.r.t. precision of the result. While the acs yielded concept descriptions that capture the information common to all input concepts completely in at least more than the half of the cases, the scs turned out to miss hardly any information on our test cases. Moreover, comparing the two approaches for obtaining good common subsumers w.r.t. a background knowledge base, it showed that the scs yields a more specific concept description than the acs in up to 80% of the cases. Since the scs showed also the best performance for computation times of the three common subsumers. To sum up, the scs seems to be an excellent alternative for the $\mathcal{ALE}(\mathcal{T})$-lcs for which we could not devise a constructive computation method in this thesis.

# Chapter 8

# Conclusions and future work

## 8.1 Discussion and conclusions

In this thesis we have investigated methods to compute non-trivial common subsumers in the presence of disjunction. The main motivation for this endeavor is the support of naive users of DL knowledge representation systems in building and maintaining their DL knowledge bases. In particular, it has been noted by users from different application domains that unbalanced concept hierarchies are an obstacle when browsing in and working with ontologies. The computation of least common subsumers is an inference service to address this problem in the following way: if the user selects a couple of sibling concepts for which she wishes to introduce a new super-concept in order to obtain a deeper structure for the concept hierarchy, such a concept can be obtained by computing the lcs of the selected concepts. The resulting concept description is then offered to the modeler to inspect, edit and add it to the knowledge base.

However, in cases where the underlying DL offers disjunction the result of the computation of the lcs is trivially the disjunction of the input concepts, which provide no insight on the commonalities shared by the selected concepts. In this thesis we investigated two approaches to remedy this problem. Each of the approaches is tailored to one application scenario to extend DL knowledge bases.

**Supporting the bottom-up approach** for extending knowledge bases, we propose to proceed in a two step procedure to obtain non-trivial common subsumers. We first 'translate' each of the selected input descriptions into a DL $\mathcal{L}_2$ that does not offer disjunction and then compute the lcs in the DL $\mathcal{L}_2$. To this end we have introduced the non-standard inference concept approximation, that realizes the translation between different DLs. In this thesis we have provided a formal definition of this new inference service, devised computation methods for concept approximation for two pairs of DLs. In particular, we have devised computation methods for $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions and for $\mathcal{ALEN}$-approximations of $\mathcal{ALCN}$-concept descriptions. Since these methods build on the computation of the lcs, the complexity of the lcs gives a lower bound for the approximation methods. More precisely, the computa-

tion of $\mathcal{ALE}$-approximations is necessarily worst case exponential, while the algorithm for computing $\mathcal{ALEN}$-approximations is at least a double-exponential time algorithm.

To assess the information loss due to first approximating the input concepts, we proposed a syntactic difference operator. Given an $\mathcal{ALC}$- and an $\mathcal{ALE}$-concept description this operator identifies the sub-concept descriptions that are captured in the $\mathcal{ALC}$-concept description alone. We have devised a heuristic algorithm to compute the difference of $\mathcal{ALC}$- by $\mathcal{ALE}$-concept descriptions.

**Supporting customization of background terminologies** requires to deal with two kind of DLs: the 'user DL' with few concept constructors and the 'background DL', where more concept constructors including disjunction are provided. We have investigated in depth the instance of this framework where the user DL is $\mathcal{ALE}$ and the background DL is $\mathcal{ALC}$. We have shown for this constellation that the lcs does not exists if the background ontology contains GCIs or is cyclic. In case the background ontology is acyclic the lcs does always exists. However, no constructive method for computing these least common subsumer could be devised. Instead we resort to compute 'good common subsumers', which need not be least. In this thesis we have described three different methods to obtain good common subsumers and two of these methods were investigated in depth in this thesis. The acs uses approximation to obtain a common subsumer w.r.t. background knowledge bases. The other method to compute good common subsumers is based on the computation of the subsumption closure is the scs. Since the method for the scs is only based on subsumption relations from the background TBox, this method can even be employed for cyclic TBoxes or TBoxes with GCIs.

In this thesis we have described our implementation of the inferences proposed here. Furthermore, we have devised a set of optimization techniques that are applicable for the computation of common subsumers as well as for approximation. Some of these techniques are already successfully applied for standard inferences, while the conjunctwise computation, for instance, is a new technique tailored to the computation of approximation.

Since all of the three algorithms to obtain non-trivial common subsumers in the presence of disjunction might not capture all the commonalities of the input *exactly*, but might lose some information, we provided an evaluation of the precision of these algorithms. The test knowledge bases used in our experiments were obtained from practical applications. The test data was chosen such that it resembles the cases encountered in the application scenario. It showed in our tests that:

- It is feasible to employ the here proposed inferences, despite their computational complexity to compute common subsumers w.r.t. TBoxes of realistic size.
- The approximation based method to compute common subsumers, does capture the exact lcs in more than the half of the cases encountered in our experiments.
- The acs performs similarly well as the approximation based approach and the scs yields even more specific common subsumers than the acs. There were only

very few cases encountered in our test suite where the scs was more general than the trivial lcs.

- The average run-times measured for the computation of the three different methods, were mostly below 1,5 seconds, with the exception of approximation, which took 6,5 seconds on the average for the larger one of our test knowledge bases.

Our implementation demonstrates that the here proposed methods can perform well enough to permit interactive use and provide a good starting point for the modeler to introduce new concept definitions.

The implementations of the non-standard inferences (NSI) investigated in this thesis are, among others, provided by our non-standard inference system Sonic. By the implementation of the different NSIs Sonic can provide reasoning support also for 'extension by import' and 'extension by modification'— besides the bottom-up approach and the customization approach.

## 8.2 Directions for future work

On the theoretical side one main direction for future work is to extend the here proposed inferences to more expressive DLs and TBox formalisms. It is clear from the results in [Baa03b], that the lcs w.r.t. cyclic TBoxes does not need to exist even for DLs only offering conjunction and existential restrictions. However, as we saw for unfoldable TBoxes, the approaches to compute good common subsumers are already helpful to provide a starting point for the modeler to come up with a new concept definition. Thus extending these approaches to general TBoxes is surely worth investigating.

For TBoxes that contain GCIs which can be treated by concept absorption, the absorption technique can be used to obtain a TBox with more concept definitions, which, if not cyclic can be treated by the methods proposed in this thesis. Furthermore, acyclic range restrictions for roles can be integrated easily into the here proposed approaches by propagating these restrictions when computing the common subsumer for the concept descriptions nested in value and existential restrictions.

Another direction to investigate is to extend the set of concept constructors for which computation methods of common subsumers are devised. In particular extending the methods from unqualified number restrictions to qualified ones seems to be an interesting choice, since the OWL 1.1 standard will include this kind of number restrictions.

This thesis itself poses open problems to be addressed in future work. For instance, a constructive method for the $\mathcal{ALE}(\mathcal{T})$-lcs (computed w.r.t. background knowledge bases), is to be devised. Such a procedure would require the structural characterization of subsumption in $\mathcal{ALC}$. This, in turn, would allow us to devise the exact difference operator, instead of just a heuristic. Equipped with such a difference operator for $\mathcal{ALC}$, we would be able to assess the information loss by computing the gcs in the customization framework.

On the practical side, we plan to complete the implementation of the here proposed optimization techniques and evaluate them. In particular, it would be interesting to

see which of the conflicting optimizations (as for instance lazy unfolding vs. conjunctwise approximation) result in a more efficient implementation. The next inference to implement in SONIC is the computation of the most specific concept (msc) or a heuristic for it to complement the bottom-up approach by deriving concept descriptions from individuals in the ABox.

The next major release of our NSI system SONIC, more precisely, the SONIC server will provide a DIG 2.0 NSI extension implementation. By the use of this extension, other programs can use the NSIs as system services. The SONIC front-end will be adapted to PROTÉGÉ 4.0, while at the time of writing a version of RACERPORTER (see [WM07]) including the SONIC reasoning services is on the way.

By the contribution of this thesis and the system SONIC developed in the course of this project, non-standard inferences are no longer only considered in theoretical respect, but can, in fact be employed to provide the reasoning services for practical applications.

# Bibliography


[AH02] M.-A. Aufaure and H. Hajji. Semantic structuration of image annotations: A data mining approach. In *Proc of the International Workshop on Multimedia Information Systems (MIS'02)*, pages 38–47, 2002. → p. 38

[Baa96] F. Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18:175–219, 1996. → p. 40

[Baa03a] F. Baader. Computing the least common subsumer in the description logic $\mathcal{EL}$ w.r.t. terminological cycles with descriptive semantics. In *Proc. of ICCS'03*, volume 2746 of *LNAI*, pages 117–130. Springer, 2003. → pp. 42, 95

[Baa03b] F. Baader. Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In G. Gottlob and T. Walsh, editors, *Proc. of IJCAI'03*, pages 325–330. Morgan Kaufmann, 2003. → pp. 42, 141

[Baa03c] F. Baader. Terminological cycles in a description logic with existential restrictions. In G. Gottlob and T. Walsh, editors, *Proc. of IJCAI'03*, pages 319–324. Morgan Kaufmann, 2003. → p. 42

[Bak95] A. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995. → pp. 4, 104

[Bat90] J. A. Bateman. Upper modeling: Organizing knowledge for natural language processing. In *Proc. of the 5th Int. Workshop on Natural Language Generation*, pages 54–61, Dawson, PA, 1990. → p. 5

[BBK01] F. Baader, S. Brandt, and R. Küsters. Matching under side conditions in description logics. In B. Nebel, editor, *Proc. of IJCAI'01*, 2001. → p. 35

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Univ. Press, 2003. → pp. 146, 148, 149, 153, 154, 155

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001. → p. 3

[BER+98] J.A. Blake, J.T. Eppig, J.E. Richardson, M.T. Davisson, and The Mouse Genome Informatics Group. The Mouse Genome Database (MGD): a community resource. Status and enhancements. *Nucleic Acids Research*, 26(1):130–137, 1998. → p. 6

[BFH+94] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994. → pp. 4, 106

[BGB+99] P.G. Baker, C.A. Goble, S. Bechhofer, N.W. Paton, R. Stevens, and A. Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999. → p. 62

[BGSS07] F. Baader, B. Ganter, U. Sattler, and B. Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *Proc. of IJCAI'07*. AAAI Press, 2007. → p. 15

[BHGS01] S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: a Reasonable Ontology Editor for the Semantic Web. In *Proc. of KI'01*, volume 2174 of *LNAI*, pages 396–408, 2001. → pp. 62, 128

[BHN+92] F. Baader, B. Hollunder, B. Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proc. of KR'92*, pages 270–281. Morgan Kaufmann, 1992. → p. 104

[BK98] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, editors, *Proc. of KI'98*, volume 1504 of *LNCS*, pages 129–140, Bremen, Germany, 1998. Springer. → p. 33

[BK00] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of KR'00*, pages 261–272, San Francisco, CA, 2000. Morgan Kaufmann Publishers. → pp. 33, 35, 55, 127

[BK05] F. Baader and R. Küsters. Non-standard inferences in description logics: The story so far. In D.M. Gabbay, S.S. Goncharov, and M. Zakharyaschev, editors, *Mathematical Problems from Applied Logic I*, volume 4 of *International Mathematical Series*, pages 1–66. Springer-Verlag, 2005. → p. 31

[BKM98] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. LTCS-Report

LTCS-98-09, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1998. See `http://lat.inf.tu-dresden.de/research/reports.html`. → p. 48

[BKM99] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proc. of IJCAI'99*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann. → pp. 12, 17, 31, 37, 41, 42, 43, 45, 46, 47, 55, 56, 67

[BKM00] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of KR'00*, pages 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers. → pp. 34, 61, 89, 126

[BKT01] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. LTCS-Report 01-06, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2001. → pp. 63, 65, 67

[BKT02a] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating $\mathcal{ALCN}$-concept descriptions. In I. Horrocks and S. Tessaris, editors, *Proc. of DL 2002*, number 53 in CEUR-WS, 2002. See `http://CEUR-WS.org/Vol-53/`. → pp. 68, 70, 123

[BKT02b] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proc. of KR'02*, San Francisco, CA, 2002. Morgan Kaufmann Publishers. → pp. 10, 14, 18, 19, 63, 123

[BL85] Ronald J. Brachman and Hector J. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann, 1985. → p. 154

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001. → p. 7

[BLM01] R. Bogusch, B. Lohmann, and W. Marquardt. Computer-aided process modeling with ModKit. *Computers & Chemical Engineering*, 25:963–995, 2001. → p. 5

[BLS06] F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proc. of IJCAR '06*, volume 4130 of *LNAI*, pages 287–291. Springer-Verlag, 2006. CEL download page: `http://lat.inf.tu-dresden.de/systems/cel/`. → p. 4

[BLS07] F. Baader, C. Lutz, and B. Suntisrivaraporn. Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In *Journal of Logic, Language and Information, Special Issue on Method for Modality (M4M)*, 2007. → p. 6

[BM00] F. Baader and R. Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, editors, *Proc. of ICCS'00*, volume 1867 of *LNAI*, pages 290–303. SV, 2000. → p. 127

[BMC03] S. Bechhofer, R. Möller, and P. Crowther. The DIG Description Logic Interface. In *Proc. of DL 2003*, Rome, Italy, 2003. → p. 128

[BMF95] J. A. Bateman, B. Magnini, and G. Fabris. The Generalized upper model knowledge base: Organization and Use. In N. Mars, editor, *Proc. of the 2nd Int. Conf. on Bulding and Sharing of Very Large-Scale Knowledge Bases*, pages 60–72, Amsterdam, 1995. IOS Press. → p. 5

[BN03] F. Baader and W. Nutt. In *[BCM+03]*, pages 43–96. Cambridge Univ. Press, 2003. → pp. 4, 25, 29

[BP07] F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. In *Proc. TABLEAUX 2007*, LNAI, Aix-en-Provence, France, 2007. Springer. → p. 10

[BPS07] F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}$. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, and A.-Y. Turhan, editors, *Proc. of DL 2007*, CEUR-WS, 2007. → p. 10

[Bra03] S. Brandt. Implementing matching in $\mathcal{ALE}$—first results. In D. Calvanese, G. De Giacomo, and E. Franconi, editors, *Proc. of DL 2003*, number 81 in CEUR-WS, 2003. See `http://CEUR-WS.org/Vol-81/`. → pp. 127, 130

[Bra06] S. Brandt. *Standard and Non-standard Reasoning in Description Logics*. PhD thesis, Institute for theoretical computer science, TU Dresden, January 2006. → pp. 15, 16, 18, 35, 39, 42, 63, 65, 67, 127, 130

[BRKM99] F. Baader, A. Borgida R. Küsters, and D. McGuinness. Matching in description logics. *J. of Logic and Computation*, 9(3):411–447, 1999. → p. 35

[BS01] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001. → p. 4

[BST04a] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In J.J. Alferes and J.A. Leite, editors, *Proc. of JELIA'04*, volume 3229 of *LNCS*, pages 400–412, Lisbon, Portugal, 2004. Springer. → pp. 14, 18, 87, 88

[BST04b] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. In V. Haarslev and R. Möller, editors, *Proc. of DL 2004*, number 104 in CEUR-WS, 2004. See `http://CEUR-WS.org/Vol-104/`. → pp. 19, 87, 88

[BST07] F. Baader, B. Sertkaya, and A.-Y. Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. of Applied Logics*, 2007. → p. 19, 87, 88, 98, 125, 135, 136

[BT01a] F. Baader and A.-Y. Turhan. TBoxes do not yield a compact representation of the least common subsumer. In C. Goble, R. Möller, and P.F. Patel-Schneider, editors, *Proc. of DL 2001*, number 49 in CEUR-WS, 2001. See `http://CEUR-WS.org/Vol-49/`. → p. 17, 45, 55

[BT01b] S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics — what does it buy me? In G. Görz, V. Haarslev, C. Lutz, and R. Möller, editors, *Proc. of ADL Workshop 2001*, number 44 in CEUR-WS, 2001. See `http://CEUR-WS.org/Vol-44/`. → p. 15, 62

[BT02a] F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In M. Jarke, J. Köhler, and G. Lakemeyer, editors, *Proc. of KI'02*, volume 2479 of *LNAI*. Springer, 2002. → pp. 17, 19, 55, 106, 122

[BT02b] S. Brandt and A.-Y. Turhan. An approach for optimized approximation. In G. Görz, V. Haarslev, C. Lutz, and R. Möller, editors, *Proc. of ADL Workshop 2002*, number 63 in CEUR-WS, 2002. See `http://CEUR-WS.org/Vol-63/`. → pp. 19, 107, 108

[BT02c] S. Brandt and A.-Y. Turhan. An approach for optimizing $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concepts. LTCS-Report 02-03, Chair f. Autom. Theory, Inst. f. Theor. C. S. TU Dresden, Germany, 2002. See `http://lat.inf.tu-dresden.de/research/reports.html`. → p. 110

[BT03] S. Brandt and A.-Y. Turhan. Computing least common subsumers for $\mathcal{FLE}^+$. In D. Calvanese, G. De Giacomo, and E. Franconi, editors, *Proc. of the 2003 International Workshop on Description Logics*, number 81 in CEUR-WS, 2003. See `http://CEUR-WS.org/Vol-81/`. → pp. 17, 41, 42

[BTK03a] S. Brandt, A.-Y. Turhan, and R. Küsters. Extensions of non-standard inferences for description logics with transitive roles. In M. Vardi and A. Voronkov, editors, *Proc. of LPAR'03*, volume 2850 of *LNCS*. Springer, 2003. → pp. 17, 41, 42

[BTK03b] S. Brandt, A.-Y. Turhan, and R. Küsters. Foundations of non-standard inferences for description logics with transitive roles. LTCS-Report 03-02, Chair f. Autom. Theory, Inst. f. Theor. C. S. TU Dresden, Germany, 2003. → p. 40

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-ref/`. → p. 7

[CAH03] R. Cornet and A. Abu-Hanna. Using description logics for managing medical terminologies. In P. Barahona M. Dojat, E. Keravnou, editor, *Artificial Intelligence in Medicine: 9th Conference on Artificial Intelligence, in Medicine in Europe (AIME 2003)*, LNCS, pages 61–70. Springer, 2003. → p. 132

[CAH04] R. Cornet and A. Abu-Hanna. Using non-primitive concept definitions for improving dl-based knowledge bases. In V. Haarslev and R. Möller, editors, *Proc. of DL 2004*, number 104 in CEUR-WS, 2004. See `http://CEUR-WS.org/Vol-104/`. → p. 132

[CBH92] W. W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In W. Swartout, editor, *Proc. of AAAI'92*, pages 754–760, San Jose, CA, 1992. AAAI Press/The MIT Press. → pp. 38, 39, 41, 61

[CG03] D. Calvanese and G. De Giacomo. Expressive description logics. In *[BCM+03]*, pages 178–218. Cambridge Univ. Press, 2003. → p. 25

[CH94a] W. W. Cohen and H. Hirsh. Learnability of description logics with equality constraints. *ML*, 17(2/3):169–200, 1994. → p. 38

[CH94b] W. W. Cohen and H. Hirsh. Learning the CLASSIC description logics: Theoretical and experimental results. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Proc. of KR'94*, pages 121–133, Bonn, 1994. Morgan Kaufmann. → p. 38

[CHKS07] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Ontology reuse: Better safe than sorry. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, and A.-Y. Turhan, editors, *Proc. of DL 2007*, pages 41–52, Italy, 2007. Bozen/Bolzano University Press. → p. 10

[CKHS07] B. Cuenca Grau, Y. Kazakov, I. Horrocks, and U. Sattler. A logical framework for modular integration of ontologies. In *Proc. of IJCAI'07*, pages 298–303, 2007. → p. 10

[Con98] The FlyBase Consortium. FlyBase: a drosophila database. *Nucleic Acids Research*, 26(1):85–88, 1998. → p. 6

[Con00] The Gene Ontology Consortium. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000. → p. 6

[CRP+93] R.A. Cote, D.J. Rothwell, J.L. Palotay, R.S. Beckett, and L. Brochu. The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL: College of American Pathologists, 1993. → p. 6

[CvHH+01] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL reference description. W3C Note, March 2001. `http://www.w3.org/TR/daml+oil-reference`. → p. 7

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962. → pp. 4, 104

[DLN+92] F. M. Donini, M. Lenzerini, D. Nardi, B. Hollunder, W. Nutt, and A. M. Spaccamela. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53:309–327, 1992. → pp. 31, 47, 119

[DLNN91] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In J. Mylopoulos and R. Reiter, editors, *Proc. of IJCAI'91*, pages 458–463, Sydney, 1991. → p. 3

[DLNN97] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134:1–58, 1997. → pp. 30, 31

[DM00] F. M. Donini and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000. → p. 31

[Don03] F. M. Donini. In *[BCM+03]*, chapter Complexity of Reasoning, pages 96–136. Cambridge Univ. Press, 2003. → p. 30

[DT02] C. Le Duc and N. Le Thanh. Approximation from a description logic with disjunction to another without disjunction. In *Proc. of ADL Workshop 2002*, CEUR-WS, Aachen, Germany, 2002. → p. 68

[DT03] C. Le Duc and N. Le Thanh. On the problems of representing least common subsumer and computing approximation in DLs. In *Proc. of DL 2003*, CEUR-WS, 2003. → p. 68

[EBBK89] D. W. Etherington, A. Borgida, R. J. Brachman, and H. A. Kautz. Vivid knowledge and tractable reasoning: preliminary report. In *Proc. of IJCAI'89*, pages 1146–1152, Detroit, US, 1989. → p. 61

[Fra03] E. Franconi. Natural language processing. In *[BCM+03]*, pages 450–461. Cambridge Univ. Press, 2003. → p. 5

[Fre95] J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, USA, 1995. → p. 104

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness*. W. H. Freeman and Company, San Francisco, 1979. → p. 42

[GLW06] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? a case for conservative extensions in description logics. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proc. of KR'06*, pages 187–197. AAAI Press, 2006. → p. 11

[GMF+03] J. Gennari, M. Musen, R. Fergerson, W. Grosso, M Crubézy, H. Eriksson, N. Noy, and S. Tu. The evolution of PROTÉGÉ-2000: An environment for knowledge-based system development. *International Journal of Human-Computer Studies*, 58:89–123, 2003. → pp. 11, 62, 128

[Gru93] T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Netherlands, 1993. Kluwer Academic Publishers. → p. 8

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin, 1999. → pp. 15, 98

[Hem01] E. Hemaspaandra. The complexity of poor man's logic. *Journal of Logic and Computation*, 11(4):609–622, 2001. → pp. 31, 50, 119

[HHK95] M. Rauch Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *IEEE Symposium on Foundations of Computer Science*, pages 453–462, 1995. → p. 42

[HLRT02a] M.-S. Hacid, A. Léger, C. Rey, and F. Toumani. Computing concept covers: a preliminary report. In I. Horrocks and S. Tessaris, editors, *Proc. of DL 2002*, CEUR-WS, Toulouse, France, 2002. "See `http://CEUR-WS.org/Vol-53/`". → pp. 38, 39

[HLRT02b] M.-S. Hacid, A. Léger, C. Rey, and F. Toumani. Dynamic discovery of e-services in a knowledge representation and reasoning context. In *Proc. of the 18th French conference on advanced databases (BDA)*, pages 21–25, Evry, France, 2002. → p. 38

[HM01a] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nebel, editor, *Proc. of IJCAI'01*, pages 161–166, 2001. → pp. 4, 39, 105

[HM01b] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkov, editors, *Proc. of IJCAR '01*, LNCS. Springer, 2001. → p. 4

[HM04] V. Haarslev and R. Möller. Optimization techniques for retrieving resources described in OWL/RDF documents: First results. In *Proc. of KR'04*, pages 163–173, 2004. → p. 4

[HMT01] V. Haarslev, R. Möller, and A.-Y. Turhan. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In R. Goré,

A. Leitsch, and T. Nipkov, editors, *Proc. of the International Joint Conference on Automated Reasoning IJCAR'01*, LNAI. Springer Verlag, 2001. → pp. 4, 104

[HMW05] V. Haarslev, R. Möller, and M. Wessel. Description logic inference technology: Lessons learned in the trenches. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proc. of DL 2005*, volume 147 of *CEUR Workshop Proceedings*, 2005. → p. 4

[Hor97] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997. → pp. 4, 104, 106

[Hor98] I. Horrocks. Using an expressive description logic: FaCT or fiction? In A.G. Cohn, L.K. Schubert, and S.C. Shapiro, editors, *Proc. of KR'98*, pages 636–647, 1998. → p. 4

[Hor02] I. Horrocks. Reasoning with expressive description logics: Theory and practice. In A. Voronkov, editor, *Proc. of CADE'02*, number 2392 in LNAI, pages 1–15. Springer, 2002. → p. 4

[HPS99] I. Horrocks and P.F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999. → pp. 4, 105

[HPSvH03] I. Horrocks, P.F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003. → p. 7

[HS05] I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of IJCAI'05*. Morgan Kaufmann, 2005. → p. 4

[HSG04] U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational Methods in Computer Science*, 1:251–276, 2004. → p. 3

[HST00] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for very expressive description logics. *J. of the Interest Group in Pure and Applied Logic*, 8(3):239–264, 2000. → p. 4

[HT00] I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of KR'00*, San Francisco, CA, 2000. Morgan Kaufmann Publishers. → p. 104

[Kat04] I. Katriel. On the algebraic complexity of set equality and inclusion. *Inf. Process. Lett.*, 92(4):175–178, 2004. → p. 120

[KB01] R. Küsters and A. Borgida. What's in an attribute? Consequences for the least common subsumer. *J. of Artificial Intelligence Research*, 14:167–203, 2001. → p. 41

[KL94] K. Knight and S. Luk. Building a large knowledge base for machine translation. In *Proc. of AAAI'94*, 1994. → p. 5

[KM00] R. Küsters and R. Molitor. Computing least common subsumers in $\mathcal{ALEN}$. LTCS-Report 00-07, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000. See `http://lat.inf.tu-dresden.de/research/reports.html`. → pp. 53, 55

[KM01a] R. Küsters and R. Molitor. Approximating most specifc concepts in description logics with existential restrictions. In F. Baader, G. Brewka, and T. Eiter, editors, *Proc. of KI'01*, volume 2174 of *LNAI*, pages 33–47. Springer, 2001. → p. 33

[KM01b] R. Küsters and R. Molitor. Computing Least Common Subsumers in $\mathcal{ALEN}$. In B. Nebel, editor, *Proc. of IJCAI'01*, pages 219–224. Morgan Kaufman, 2001. → p. 17, 37, 41, 42, 48, 49, 50, 52, 53, 54, 71, 78

[KM02] R. Küsters and R. Molitor. Approximating most specific concepts in description logics with existential restrictions. *AI Communications*, 15(1):47–59, 2002. → p. 33

[KMR04] H. Knublauch, M. Musen, and A. Rector. Editing description logic ontologies with the PROTÉGÉ OWL plugin. In V. Haarslev and R. Möller, editors, *Proc. of DL 2004*, number 104 in CEUR-WS, 2004. See `http://CEUR-WS.org/Vol-104/`. → pp. 11, 128

[KPS+06] A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca Grau, and J.A. Hendler. Swoop: A web ontology editing browser. *J. Web Sem.*, 4(2):144–153, 2006. → p. 10

[KRM07] M. C. Keet, M. Roos, and M. S. Marshall. A survey of requirements for automated reasoning services for bio-ontologies in OWL. In *Proc. of Third international Workshop OWL: Experiences and Directions (OWLED 2007)*, 2007. → p. 12

[Küs98] R. Küsters. Characterizing the semantics of terminological cycles in $\mathcal{ALN}$ using finite automata. In *Proc. of KR'98*, pages 499–510, 1998. → p. 40

[Küs01] R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *LNAI*. Springer, 2001. Ph.D. thesis. → pp. 35, 39, 41, 42, 47, 62, 79, 80, 83

[LB87] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987. → p. 3

[LSW01] C. Lutz, U. Sattler, and F. Wolter. Description logics and the two-variable fragment. In D.L. McGuiness, P.F. Pater-Schneider, C. Goble, and R. Möller, editors, *Proc. of the 2001 International Workshop in*

*Description Logics (DL-2001)*, pages 66–75, Stanford, California, USA, 2001. → p. 3

[Lut99] C. Lutz. Complexity of terminological reasoning revisited. In *Proc. of LPAR'99*, LNCS, pages 181–200. Springer, 1999. → p. 31

[LWW07] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proc. of IJCAI'07*. AAAI Press, 2007. → p. 11

[Man01] Th. Mantay. *Commonality-based information retrieval with a terminological knowledge representation system*. PhD thesis, Department of Computer Science, Univ. of Hamburg, Germany, 2001. → p. 41

[McG96] D. L. McGuinness. *Explaining Reasoning in Description Logics*. PhD thesis, Rutgers University, 1996. → pp. 34, 61

[MH03] R. Möller and V. Haarslev. Description logic systems. In *[BCM+03]*, pages 282–305. Cambridge Univ. Press, 2003. → p. 38

[MHN98] R. Möller, V. Haarslev, and B. Neumann. Semantics-based information retrieval. In *Proc. IT&KNOWS-98: International Conference on Information Technology and Knowledge Systems*, pages 49–6, Vienna, Budapest, 1998. → p. 38

[Min74] M. Minsky. A framework for representing knowledge. Technical report, MIT-AI Laboratory, Cambridge, MA, USA, 1974. → p. 2

[MM98] Th. Mantay and R. Möller. Content-based information retrieval by computation of least common subsumers in a probabilistic description logic. In *Proc. International Workshop on Intelligent Information Integration, ECAI'98, Brighton UK, 1998,*, 1998. → p. 38

[Mol00] R. Molitor. *Unterstützung der Modellierung verfahrenstechnischers Prozesse durch Nicht-Standardinferenzen in Beschreibungslogiken*. PhD thesis, Department of Computer Science, RWTH Aachen, Germany, 2000. In German. → p. 12

[Mot06] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe, 2006. → p. 4

[MSH07] B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In F. Pfennig, editor, *Proc. of CADE'07*, LNAI, pages 67–83, Bremen, Germany, 2007. Springer. → p. 4

[MYM07] J. Morbach, A. Yang, and W. Marquardt. OntoCAPE—A large-scale ontology for chemical process engineering. *Engineering Applications of Artificial Intelligence*, 20(2):147–161, 2007. → pp. 5, 6, 132

[Neb88] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988. → pp. 3, 44

[Neb90] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990. → pp. 4, 31

[Pap94] Ch. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., Reading, Massachussetts, 1994. → p. 30

[PMB+91] P. Patel-Schneider, D. L. McGuinness, R. J. Brachman, L. Alperin Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3):108–113, 1991. → p. 38

[PS93] P.F. Patel-Schneider and Bill Swartout. Working version (draft): Description logic specification from the KRSS effort, June 1993. Unpublished Manuscript. → p. 122

[PSK05] B. Parsia, E. Sirin, and A. Kalyanpur. Debugging OWL Ontologies. In *Proc. of the 14th Int. World Wide Web Conference (WWW2005)*, pages 633–640, Japan, 2005. → p. 10

[Qui67] M. Ross Quillian. Word concepts: A theory and simulation of some basic capabilities. *Behavioral Science*, 12:410–430, 1967. Republished in [BL85]. → p. 2

[Rac05] Racer Systems GmbH & Co. KG. *RacerPro Reference Manual Version 1.9*, Dec. 2005. Available from: `http://www.racer-systems.com/products/racerpro/reference-manual-1-9.pdf`. → pp. 4, 104, 128

[Rec03] A. Rector. Medical informatics. In *[BCM+03]*, pages 406–426. Cambridge Univ. Press, 2003. → p. 6

[Res] AT&T Labs Research. Grappa—a java graph package. Web site. `http://www.research.att.com/~john/Grappa/`. → p. 130

[RNG93] A. Rector, W. Nowlan, and A. Glowinski. Goals for concept representation in the GALEN project. In *Proc. of the 17th annual Symposium on Computer Applications in Medical Care, Washington, USA, SCAMC*, pages 414–418, 1993. → p. 6

[Sat98] U. Sattler. *Terminological knowledge representation systems in a process engineering application*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998. → p. 6

[Sch91] K. Schild. A correspondence theory for terminological logics: preliminary report. In J. Mylopoulos and R. Reiter, editors, *Proc. of IJCAI'91*, Sydney, Australia, 1991. → p. 3

[SCM03] U. Sattler, D. Calvanese, and R. Molitor. In *[BCM+03]*, chapter Relationships with other Formalisms, pages 137–177. Cambridge Univ. Press, 2003. → p. 3

[Ser07] B. Sertkaya. *Formal Concept Analysis Methods for Description Logics*. PhD thesis, Chair for automata theory, TU Dresden, 2007. → pp. 15, 16, 98, 103, 125, 135

[Smo88] G. Smolka. A feature logic with subsorts. Technical Report 33, IWBS, IBM Deutschland, P.O. Box 80 08 80 D-7000 Stuttgart 80, Germany, 1988. → p. 29

[Sow91] J. F. Sowa, editor. *Principles of Semantic Networks*. Morgan Kaufmann, 1991. → p. 2

[Sow92] J. F. Sowa. *Encyclopedia of Artificial Intelligence*, chapter Semantic Networks. John Wiley & Sons, New York, 1992. → p. 2

[SP04] E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. In V. Haarslev and R. Möller, editors, *Proc. of DL 2004*, volume 104 of *CEUR Workshop Proceedings*, 2004. → pp. 4, 104

[SS88] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with unions and complements. Technical Report SR-88-21, DFKI, Kaiserslautern, Germany, 1988. → pp. 31, 119

[SS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991. → pp. 4, 91

[TB07] A.-Y. Turhan and Y. Bong. Speeding up approximation with nicer concepts. In D. Calvanese, E. Franconi, V. Haarslev, D. Lembo, B. Motik, S. Tessaris, and A.-Y. Turhan, editors, *Proc. of DL 2007*, 2007. → pp. 19, 137

[TBK+06] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithöner. DIG 2.0 – Towards a flexible interface for description logic reasoners. In B. Cuenca Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *In Proc. of the second international workshop OWL: Experiences and Directions*, 2006. → pp. 20, 128

[Tee94] G. Teege. Making the difference: A subtraction operation for description logics. In P. Torasso, J. Doyle, and E. Sandewall, editors, *Proc. of KR'94*, pages 540–550, Bonn, FRG, 1994. Morgan Kaufmann. → pp. 39, 78

[TH06] D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of IJCAR '06*, 2006. FaCT++ download page: `http://owl.man.ac.uk/factplusplus/`. → pp. 4, 104

[THPS07] D. Tsarkov, I. Horrocks, and P. F. Patel-Schneider. Optimising terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 2007. → pp. 4, 104, 105

[TK04a] A.-Y. Turhan and C. Kissig. SONIC — Non-standard inferences go OILED. In D. Basin and M. Rusinowitch, editors, *Proc. of IJCAR '04*, volume 3097 of *LNCS*, pages 321–325. Springer, 2004. SONIC is available from `http://wwwtcs.inf.tu-dresden.de/~sonic/`. → pp. 20, 122, 123, 128

[TK04b] A.-Y. Turhan and C. Kissig. SONIC—System description. In V. Haarslev and R. Möller, editors, *Proc. of DL 2004*, number 104 in CEUR-WS, 2004. See `http://CEUR-WS.org/Vol-104/`. → pp. 20, 122, 123, 128

[TM01] A.-Y. Turhan and R. Molitor. Using lazy unfolding for the computation of least common subsumers. In C. Goble, R. Möller, and P.F. Patel-Schneider, editors, *Proc. of DL 2001*, number 49 in CEUR-WS, 2001. See `http://CEUR-WS.org/Vol-49/`. → pp. 19, 122

[Tob01] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001. → pp. 4, 31

[Tur05] A.-Y. Turhan. Pushing the SONIC border — SONIC 1.0. In Reinhold Letz, editor, *Proc. of Fifth International Workshop on First-Order Theorem Proving (FTP 2005)*. Technical Report University of Koblenz, 2005. http://www.uni-koblenz.de/fb4/publikationen/gelbereihe/RR-13-2005.pdf. → pp. 19, 20, 128

[Vai84] L. G. Vailant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134 – 1142, 1984. → p. 38

[WBH+05] K. Wolstencroft, A. Brass, I. Horrocks, P. Lord, U. Sattler, R. Stevens, and D. Turi. A little semantic web goes a long way in biology. In *Proc. of the 2005 International Semantic Web Conference (ISWC 2005)*, number 3729 in LNCS, pages 786–800. Springer, 2005. → p. 16

[WLT+06] K. Wolstencroft, P. W. Lord, L. Tabernero, A. Brass, and R. Stevens. Protein classification using ontology classification. In *In Proc. 14th International Conference on Intelligent Systems for Molecular Biology ISMB'06 (Supplement of Bioinformatics)*, pages 530–538, 2006. → p. 6

[WM07] M. Wessel and R. Möller. Design principles and realization techniques for user friendly, interactive, and scalable ontology browsing and inspection tools. In *International Workshop on OWL: Experiences and Directions (OWLED 2007)*, 2007. → p. 142